METHODS FOR HIGH DIMENSIONAL MATRIX COMPUTATION AND DIAGNOSTICS OF DISTRIBUTED SYSTEM

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Wei Chen

May 2014

METHODS FOR HIGH DIMENSIONAL MATRIX COMPUTATION AND DIAGNOSTICS OF DISTRIBUTED SYSTEM

Wei Chen, Ph.D.

Cornell University 2014

Big data provides opportunities, but also brings new challenges to modern scientific computing. In this thesis, we conduct sparse principal component analysis (SPCA) on high dimensional matrices. We propose a modified curvilinear algorithm to solve eigenvalue optimization with orthogonal constraints, and combine it with an augmented Lagrangian method to improve its computational efficiency. We compare our algorithm against standard PCA on the recovery of low-rank tensors and a mean-reverted statistical arbitrage strategy. The explosion of big data has also influenced the development on distributed computing systems. For debugging purposes, we are interested in predicting server run-time based on system data early in the process. We study discriminative models in functional data analysis, and introduce generative models that capture server regime-change behaviors. We also design computational methods, including a blocked Gibbs sampler, to improve the accuracy and efficiency of model estimation.

BIOGRAPHICAL SKETCH

Wei Chen was born in Fuzhou, China on December 28, 1984. At the age of fifteen, he entered the talented class of Fuzhou No.3 Middle School. After high school, he moved to Hangzhou to pursue advanced studies at Chu Konchen Honors College of Zhejiang University, where he obtained his Bachelor's degree with distinction in Mathematics. Upon his graduation from Zhejiang University, Wei decided to continue his education, and joined the Department of Applied Mathematics and Statistics of Johns Hopkins University, where he obtained his Master's degree with distinction in Applied Mathematics.

After his graduation, Wei joined the School of Operations Research and Information Engineering at Cornell University as a Ph.D. student. Under the guidance of his advisor Martin Wells, he completed his dissertation in the field of Applied Statistics, with a focus on algorithms about sparse principal component analysis, functional data analysis and probabilistic graphical model. During his PhD study, Wei applied his knowledge and skills to various practical problems in the industry, and traveled around the world to collaborate with brilliant professionals from different countries. He interned at Microsoft Research (Mountain View), Nomura Securities (Tokyo), and Goldman Sachs (London) to work on challenging quantitative problems.

Upon graduation from Cornell University with his Ph.D., Wei is excited to pursue his dream in the financial industry, where he has the opportunity to leverage his math, statistics and programming skills.

To my parents:

Maoxin Chen and Yanyan Xu

To my wife:

Jing Xie

ACKNOWLEDGEMENTS

This thesis could not have been completed without the help and support from many people around me.

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Martin Wells, for his patient guidance and consistent support throughout my five-year Ph.D. study. He led me to a fascinating realm where statistical modeling and big data applications meet each other. He gave me total freedom to explore my research interest, and offered innovative ideas and warmhearted help on my research and career development. It is a great honor and privilege for me to work closely and learn from him.

I am very grateful to the other two members in my committee, Prof. Bruce Turnbull and Prof. Ao Tang for their helpful comments and suggestions on my work. I would like to thank the entire faculty at the School of Operations Research and Information Engineering. Their courses and talks broaden my academic horizon and enrich my knowledge.

I would like to acknowledge my supervisor from Microsoft Research, Moises Goldszmidt, for his interesting computing system project, which becomes a part of my thesis. I would also like to express my thanks to my previous colleagues, Dahua Lin, Qilong Zhang and Paul Kondratko for their career suggestions. I also want to thank Liyan Jia, Yu Zhe, Chaoxu Tong, Cao Ni, Jiayang Gao, Zhengqiang Tang, Liaoruo Wang, and many other friends and students at Cornell University for making my life at Ithaca memorable.

Finally, I would like to thank my parents, Maoxin Chen and Yanyan Xu, for their endless and unconditional love and support; my wife Jing Xie, who stayed with me over the last five years. Her consistent encouragement and advice helped me to challenge tough questions. Without her, I may have given up my Ph.D. study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

According to McKinsey's Business Technology Office (MBT), the ability of analyzing so-called "big data" will become a key basis of competition, and a basis for new waves of productivity, growth and innovation. What was the domain of companies like Microsoft, Google, Yahoo, Facebook, and Amazon, namely, crunching through terabytes to petabytes of data for services such as search, social networks, customer preferences, and news portals, up to about 5 years ago, is becoming the focus of more and more companies on all sectors including government. MBT estimates that by 2009 nearly all sectors in the US economy had at least an average of 200 terabytes of stored data (twice the size of Wal-Mart's data warehouse in 1999) per company with more than 1,000 employees. This data includes business intelligence data, (e.g. consumer baskets and habits, marketing data, etc), and data about the processes in the companies themselves, and data about sentiment on companies products and services gathered from the web and the social networks themselves.

1.1 Matrix Computation Problem

The invention of internet moves us from the traditional text-based communication to the new interactive media, while improvements in data storage technology makes it possible to save enormous amount of data directly or indirectly from users, markets, or even environment (Michael and Miller 2013). Big data can help us explore people's behavior patterns. For instance, some social network companies and electronic commerce companies can analyze customers' characteristics from their social connections or shopping records, and predict their interests and decision making processes; big data can also pro-

vide us some valuable insights to the hidden disciplines of the markets. For instance, some algorithmic trading companies can trade against some statistical mispriced securities to make profits if they study price relationships between thousands of securities in the market; big data can even lead to discovery of prevention or cures for some diseases. The Human Genome Project and Human Brain Project are typical bioinformatic efforts researchers have done so far. However, big data brings not only opportunities, but also challenges to modern data analysis, which requires robust and efficient analysis approaches. Fan et al. (2013) pointed out that big data has two important characteristics: high dimensionality and large sample size. The challenges raised by these two features include noise accumulation, spurious correlations, homogeneity, heterogeneity, and computational costs. Many traditional methods perform well in low dimensional or moderate sample size, but do not scale to high dimensional or large sample size.

To face the challenges of big data, technology about dimension reduction and variable selection has been extensively studied and employed in data analysis. For instance, various regularization and variable selection methods are developed over the last few decades (Fan et al. 2013), including shrinkage via lasso (Tibshirani 1996), signal decomposition via basis pursuit (Chen et al. 1998), nonconcave penalized likelihood (Fan and Li 2001), Dantzig selector (Candes and Tao 2007), minimax concave penalty (Zhang et al. 2010) and etc. We specifically consider principal component analysis (PCA) (Jolliffe 1986), which is one of the most popular dimension reduction techniques and has been widely used in scientific applications, e.g., image recognition (Hancock et al. 1996), gene expression (Alter et al. 2000), natural language processing (Hastie et al. 2005), and etc. Variations of PCA include functional PCA (Friston et al. 1993), nonlinear PCA (Schölkopf et al. 1998), probabilistic PCA (Tipping and Bishop 1999), multinomial PCA (Buntine 2002), kernel PCA (Scholkopf et al. 1999), generalized PCA (Vidal et al. 2005), sparse PCA (Zou et al. 2006) and etc.

Sparse principal component analysis (SPCA) is one of the most efficient PCA methods under the big data scenario. The main drawback of PCA is the low interpretability of principal components (PCs), especially when we apply PCA to high dimensional data. SPCA fixes this issue by generating principal components with a lot of zero loadings. There is substantial work on SPCA over the last few decades. The first class of SPCA is the heuristic modification of PCs from the standard PCA, e.g., factors rotation (Jolliffe 1995), artificial threshold of eigenvectors (Cadima and Jolliffe 1995); the second class is the optimization formulation of SPCA, e.g., LASSO based PCA (Jolliffe et al. 2003), nonconvex approximation (Zou et al. 2006, Sriperumbudur et al. 2007); the third class is the spectral analysis and singular value decomposition (Moghaddam et al. 2006, 2007, Shen and Huang 2008); and the fourth class is semidefinite programming (SDP) (D'Aspremont et al. 2007, d'Aspremont et al. 2008, Zhang and El Ghaoui 2011). For other theoretical work on SPCA, see Journée et al. (2010), Amini and Wainwright (2008), Yuan and Zhang (2011), Asteris et al. (2011).

Although the above SPCA methods eventually obtain sparse PCs, they do not take the correlation of PCs into account – some methods do not even consider the orthogonality of loading vectors. Moreover, these methods over maximize the total explained variance of sparse PCs. Lu and Zhang (2012) pointed out these two issues and proposed their formulation of SPCA:

$$\begin{aligned} \max_{X \in R^{n \times p}} \; & tr(X^T \Sigma X) - \rho |X| \\ s.t. \quad & |X_i^T \Sigma X_j| \leq \Delta_{i,j}, \\ & X^T X = I, \end{aligned} \tag{1.1}$$

where $\Delta_{i,j} \geq 0 (i \neq j)$ are tuning parameters for the correlation of X, and $\rho > 0$ is the penalized parameter of sparsity. This formulation improves upon the other SPCA methods by introducing additional constraints. To solve (1.1), Lu and Zhang (2012) introduced

an augmented Lagrangian algorithm, and proved its convergence under certain assumptions. They also mentioned the importance of finding an starting point when applying the algorithm to high-dimensional matrices.

In this thesis, we propose an algorithm for finding a feasible starting point. Instead of choosing a starting point randomly (Lu and Zhang 2012), we solve a generalized problem of (1.1)

$$\max_{X \in R^{n \times p}} tr(X^T \Sigma X) \quad s.t. \quad X^T X = I, \tag{1.2}$$

which can be classified as a manifold optimization problem, since the feasible set $\{X : X^T X = I\}$ of (1.2) is Stiefel manifold. A variety of algorithms have been proposed during the last few decades for manifold optimization, including retraction algorithms (Adler et al. 2002, Absil et al. 2007, Baker et al. 2008, Absil et al. 2009), steepest descent gradient (Helmke et al. 1994, Udriste 1994), conjugate gradient (Edelman et al. 1998, Qi et al. 2010) and Newton's method (Smith 1994, Owren and Welfert 2000, Smith 2013). These algorithms typically preserve the manifold constraints during the iterations.

In recent literature, Wen and Yin (2013) proposed a curvilinear algorithm to solve (1.2). We improve their algorithm in the computational efficiency of the large matrices computation, and incorporate it with the augmented Lagrangian algorithm. We compare the hybrid algorithm against other existing SPCA methods on randomly generated matrices. Our results show that our hybrid algorithm is computationally more efficient than the existing methods.

In terms of application, we apply our algorithm to a mean-reverted statistical arbitrage strategy (Avellaneda and Lee 2010), and show its performance on trading signals of *S&P* 500 from 2007 to 2013. Statistical arbitrage is a variety of trading and investment strategies, and has been extensively studied since 1990s (Poterba and Summers

1988, Lehmann 1990, Lo and MacKinlay 1990, Admanti and Pfleiderer 1991, Barclay and Warner 1993, Miller et al. 1994, Chan and Lakonishok 1993, 1995, Bollerslev and Ole Mikkelsen 1996, Davis et al. 1997, Lo 1999, Bookstaber 2000, Lo 2001, Cont et al. 2002, Boss et al. 2004, Andersen et al. 2006, Khandania and Lob 2007, Avellaneda and Lee 2010).

In a generic statistical arbitrage strategy, investors create a market-neutral portfolio with low volatility, by pairing up a large number of stocks to diversify the risk. We particularly consider a mean-reverted strategy. In this strategy, we decompose the stock return into two parts, a systematic part and an idiosyncratic part. We model the systematic part by regressing it on some other reference returns (called "market factors"), and model the idiosyncratic part by a mean reversion process. Statistical arbitragers believe that if market factors are chosen wisely to diversify the systematic risk, the idiosyncratic return will oscillate around its long-run mean. This provides an opportunity for them to lock their profit by shorting the stock when the idiosyncratic return is above the long-run mean, and longing the stock when the idiosyncratic return is below the long-run mean. Avellaneda and Lee (2010) studied two methods for generating market factors, mainly, standard PCA and exchange-traded funds (ETFs), based on trading signals of *S&P* 500 from 2003 to 2007. Although they are easy to implement, issues remain:

1. Derived portfolios from PCA are difficult to interpret.
2. Trading each constituent of portfolios from PCA takes extra transaction cost.
3. ETFs contain correlated information.
4. ETFs requires prior information about the economy and the market.

In Chapter 2, we use SPCA to generate market factors, and compare its performance against PCA and ETFs. We show that SPCA simplifies the components of derived

portfolios, reduces the transaction costs, and performs uniformly better than PCA and ETFs.

We also apply our hybrid algorithm to low-rank tensor estimation. Tensor, also known as multi-dimensional array, is used to represent relationships between sets of geometric vectors. It has been widely used in physics (Danielson and Danielson 1997, Chaikin and Lubensky 2000, Jeevanjee 2011), psychometrics (Grieve et al. 2007, Kodl et al. 2007), chemometrics (Kolda and Bader 2009, Lim and Comon 2009), signal processing (De Lathauwer and De Moor 1998, De Lathauwer et al. 2000a, Westin et al. 2002), computer vision (Hartley and Zisserman 2003, Medioni and Kang 2004, Shashua and Hazan 2005, Aja-Fernández 2009), neuroscience (Martınez-Montes et al. 2004, Miwakeichi et al. 2004, Damoiseaux et al. 2006), and elsewhere. Under the big data scenario, decomposition and interpretation of tensors have become more and more important. Many tensor decomposition methods have been proposed over the last few decades, like INDSCAL (Carroll and Chang 1970), PARAFAC2 (Harshman 1972), DEDICOM (Harshman 1978), CANDELINC (Carroll et al. 1980), PARATUCK2 (Harshman and Lundy 1996). Among all, CANDECOMP/PARAFAC (Carroll and Chang 1970, Harshman 1970) and Tucker (Tucker 1966, De Lathauwer et al. 2000b) are the most popular ones nowadays. However, these methods use non-convex optimization for estimating tensors, which suffers certain convergence issues.

Recently, convex optimization has been introduced to the estimation of two–dimensional tensor (Fazel et al. 2001, Srebro et al. 2004, Evgeniou and Pontil 2007, Tomioka and Aihara 2007), with development of theory on matrix reconstruction (Candès and Recht 2009, Recht et al. 2010). The estimation has also been generalized to tensors with higher dimensions (De Silva and Lim 2008, Signoretto et al. 2010, Gandy et al. 2011, Liu et al. 2013, Rauhut et al. 2013). Tomioka et al. (2010) proposed

three convex formulations for the low-rank tensor reconstruction, and solved their problems by alternating direction method of multipliers (ADMM) (Gabay and Mercier 1976, Boyd et al. 2011). In Section 2.2, we apply our hybrid algorithm to ADMM, show that the modified algorithm performs roughly 25% faster than ADMM.

Other optimization work about large-scale computing include random sampling in convex programming (Bertsimas and Vempala 2004), mixed integer optimization on least quantile of squares (Bertsimas and Mazumder 2013), mixed integer programming with automated configuration (Hutter et al. 2010), Global mixed-integer quadratic optimizer (Misener and Floudas 2013), robust optimization (Bertsimas et al. 2011), interior point method with warm-start point (Colombo et al. 2011), interior point method with l_1-regularized least squares (Kim et al. 2007), and some optimization software for large scientific programming, like CPLEX (Cplex 2007, CPLEX 2009) and Gurobi (Optimization 2012) are also developed and widely used in both academia and industry.

1.2 Computation System Problem

The explosion of the availability of data has also influenced a parallel development on the computing infrastructure. This infrastructure consists of software, which takes a linear program, distributes and parallelizes its application into large clusters of machines. These platforms include Google's Map/Reduce, Yahoo's Hadoop, and Microsoft Dryad (Isard et al. 2007, Yu et al. 2008). Estimates vary but these platforms are widely used with Yahoo running more than 40,000 nodes of Hadoop with their biggest single cluster now at 4,500 servers. Facebook runs a 1,100 node cluster and a second 300 node cluster. LinkedIn runs many clusters including deployments of 1,200,580, and 120 nodes. What has not been keeping with this rapid development are tools and methods for

debugging the performance of these systems. This thesis starts to address this problem in a statistically sound approach.

In general, these computing platforms are (loosely) based on the map/reduce model, first reported in the literature (in the context of big data) in Dean and Ghemawat (2008). Very loosely, in this model the computation proceeds in stages where each stage is either a *map* stage or a *reduce* stage. Each stage is composed of a set of nodes performing the same computation in parallel on different partitions of the data. The *map* stage executes the same operation (may be a complex piece of code) in all nodes and each node in the *reduce* stage consumes the output of various nodes in the map stage and so on. As all nodes in a particular stage (either map or reduce) perform the same operation, it is expected that their running time should be similar. In practice, this is not always the case. The problem is that a single outlier may destroy the inherent parallelism, as for example the whole reduce stage will not start until the preceding map stage finishes. Thus the running time of the whole program, suddenly depends on a single outlier. Figure 1.1 shows this phenomena on a real computation. This program applies machine learning algorithms to analyze clicking behavior in order to detect clicking-robots. The horizontal axis is time (in fractions of hours) and the vertical axis are physical machines. A horizontal line represents the running time of a machine. Note that a handful of outliers, approximately 15 out of 250 machines, cause total running time of the stage to extend from 30 minutes to one hour and 30 minutes.

There are many possible causes of this misbehavior: (a) it can be that the initial partition of the data is not balanced and some nodes consume more data than others (hence the computation takes longer); (b) It can be that the node is not reading the data from a local disk but over the network, or from a congested disk; (c) it can be that the node is faulty (there is a hardware problem). In any case an early detection of an outlier

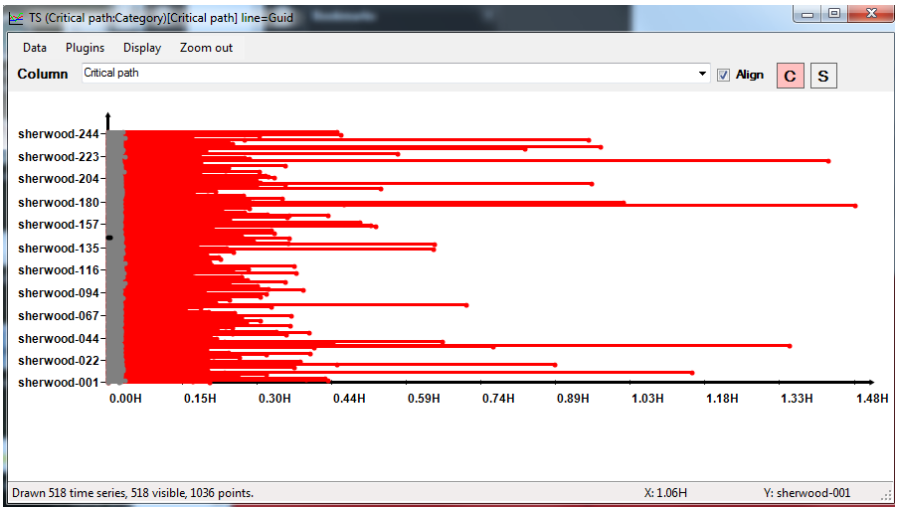

Figure 1.1: Variance in running time for a given stage in a program for detecting ClickBots. Notice that a few vertices (horizontal lines) are responsible for most of the waiting time.

is beneficial for various reasons: First, we can use it to estimate the running time of the whole program. Second, we can alert the job manager of the situation so that diagnostics can be executed on the node to determine whether it is faulty. If it is the case that the node is faulty, then a duplicate node can be activated to continue with the job. Note that duplication is not always a beneficial action. For example, if the slowness of the node is due to the fact that it is reading from a physical disk that is congested (because too many other vertices are sharing it) duplicating the node will only exacerbate the problem. It may be possible to execute other more complicated actions such as on the fly reallocation of computational nodes (to balance the load on disks), or change the dependency between the different nodes in the different stages.

In this thesis, we introduce the classification models to predict whether the server processing time will be normal based on server metrics. The modeling is inspired by two different methodologies, discriminative and generative methods. In a discriminative method, a parameter model is introduced to compute the mappings between latent variables and observed variables directly, and the values of parameters are inferred from

the training data; while in a generative method, a joint distribution of both latent and observed variables is proposed and estimated. This can be done, for instance, by learning the priors of latent variables and conditional distributions of observed variables separately, and obtaining the posterior distribution by Bayes theorem.

We provide two different models, one from each method. The first one is functional linear model, a type of functional data analysis approach. It is a distriminative method, since we link the running time of servers and its metrics by logistic functional regression. Functional data analysis (FDA) is an effective approach for analyzing data from curves and surfaces, and has recently received substantial interest in the statistics literature. It has also been applied to scientific and industrial settings extensively. Specifically we use methods for regression with functional predictors and a scalar outcome (Ramsay 2006, Ramsay and Silverman 2002). The predictors in our context are the time series observations of the server metrics, while the outcome is either the real-valued run time or a binary indicator of whether the run time is above a specified tolerance. Instead of treating the time series observations as separate predictors, the FDA approach treats them as noisy, discrete observations of unknown continuous functions. These functions are related to the scalar outcome via a model such as the functional linear model (Cardota et al. 1999).

This FDA approach has several advantages for prediction and interpretation. First, since the time series structure of the predictors is built into the model, the signal and noise in that time series predictor can be distinguished more accurately. Effectively the time series predictors are smoothed over time, so that information from the whole time series is used to remove the noise in each observation. This improves predictive accuracy of the regression model. Second, the FDA model relating the functional predictor to the outcome can be much more interpretable than the output of a simpler regression model.

For instance the functional linear model relates the outcome W to a predictor function $X(t)$ via

$$g(EW) = \alpha + \int_{\mathscr{C}} X(t)\,\beta(t)\,dt, \tag{1.3}$$

where g is referred to as the "link" function, α is the intercept, W is a real random response, EW is the expected value of W, and $X(t)$, $\beta(t)$ are square integrable random functions defined on some compact set $\mathscr{C}$ of $\mathscr{R}$. The coefficient function $\beta(t)$ is smoothly estimated, and the sign and magnitude of $\beta(t)$ over time indicates the relationship between $X(t)$ and EW.

The method of Goldsmith et al. (2010) is based on the functional linear model. We apply this method, as well as several related approaches, to the problem of predicting server run-time in a commercial computing system (Microsoft's DryadLINQ (Isard et al. 2007, Yu et al. 2008)). We compare their performance to that of an additive generalized linear model that uses the time series observations as separate predictors. Our results show that the FDA methods have better classification accuracy when predicting the binary indicator of whether the run time exceeds a specified tolerance. A cost analysis shows that this yields up to a 20% lower cost associated with classification errors. For predicting the continuous run time, the method of Goldsmith et al. (2010) has roughly 5% lower errors. We also find that the FDA methods yield far more interpretable results. These results show the value added by using FDA methods in computing system management.

The other model is Hidden Markov/Semi-Markov Regime-Change Autoregression model (HMRCA/HSMRCA), which is based on hidden Markov/semi-Markov model except that it has multiple regimes and allows regime switching for sample series, and the mean of regimes follow autoregression process. This novel model is proposed regarding that the characteristics of the distributed computing data are not captured ef-

fectively using the linear model approach. The reason is that each server goes through phases of accessing data, doing computation, waiting for sub-task completion by external systems, etc., and that the duration of each phase is stochastic. Incorporating regime-change behavior into this new model will allow us to simulate the different phases each server goes through. This novel generative model can capture some behaviors that some discriminative models cannot, for instance, differences in variability of the time series for different values of the outcomes. Besides that, we also design and implement a blocked Gibbs sampling to draw sample series directly from the posterior distributions of time series parameters in Hidden Markov Regime-Change Autoregression model (HMRCA).

Hidden Markov model (HMM), as one of the most famous generative probabilistic model, is widely used in different research areas, like signal recognition (Juang and Rabiner 1991, Andrieu and Doucet 1999), computational biology (Leroux and Puterman 1992, Krogh et al. 1994), genomics (Churchill 1992, Liu et al. 1999), image processing (Choi et al. 2000), econometrics (Hamilton 1989, 1990, Albert and Chib 1993) and elsewhere. HMM assumes that there is a set of states $\mathscr{S} = \{1, 2, \ldots, S\}$ and the associated distributions $\{\mathscr{F}_i\}$ for each state $i = 1, \cdots, S$. The time series observations $\{X_t\}$, $t = 1, \ldots, T$, depend on their unobserved hidden states $\{Z_t\}$, $t = 1, \ldots, T$, which follow a Markov chain on $\mathscr{S}$ with transition matrix $\boldsymbol{Q} = \{Q_{i,j}\}$, $i, j = 1, \ldots, S$.

$$P(Z_t \mid Z_{t-1}) = Q_{Z_{t-1}, Z_t} \quad \text{and} \quad P(X_t \mid Z_t) = \mathscr{F}_{Z_t}(X_t). \tag{1.4}$$

As an extension of HMM, hidden semi-Markov model (HSMM) allows the underlying stochastic process of $\{Z_t\}$, $t = 1, \cdots, T$, to be semi-Markov chain with sojourn time. The main difference between HMM and HSMM is that HMM only allows one observation for each state while each state can generate a sequence of observations in HSMM (Yu 2010, Si et al. 2011).

The regime-switching idea has existed for a long time in econometric community (Van Norden and Vigfusson 1996, Piger 2009). From time to time, economic data exhibit dramatic changes, associated with short-term events like financial crises, changes in government policies or long-term events like economic recessions. Consequences of those changes are described by different regimes. In early literature of econometrics, the underlying model for each regime is the same, such as $AR(1)$, and the regime-switching time follows a Markov chain; while in more complex models, high dependencies among observations are considered, and the coefficients are also subject to changes in regimes (Hamilton 2005, Kim and Nelson 1999). In Chapter 5, we have similar regime-switching properties in HMRCA/HSMRCA. We differ from the above literature in two main aspects: (1) the underlying parameters are also assumed to follow some processes. (2) priors for parameters are chosen carefully to avoid serious problems with hierarchical models.

We compare the HMRCA/HSMRCA against a parsimonious model without regime-change property, Hidden Markov Autoregression (HMA), and find that the regime-change design improves the prediction accuracy by reducing 20% of the classification errors and 30% of the associated costs. We also find that HMRCA/HSMRCA obtain similar accuracy with PFR, and both models are highly recommended in computing system diagnostics.

1.3 Thesis Organization

In this section, we summarize the contents of each chapter.

Chapter 2 considers the problem of SPCA in low-rank matrices. We first discuss a convex optimization problem with orthogonal constraints, and propose a modified curvi-

linear algorithm to solve it. We then develop an hybrid algorithm by incorporating the modified curvilinear algorithm into an augmented Lagrangian algorithm, and show that our hybrid algorithm is computationally better than the original augmented Lagrangian algorithm. We also consider a mean-reverted statistical arbitrage strategy, and apply our hybrid algorithm to generate market factors for the strategy. We backtest the strategy on historical data of *S&P* 500, and show that SPCA performs better than PCA and ETFs methods with lower transaction costs and more interpretable portfolios.

Chapter 3 addresses the low-rank tensor estimation. We discuss three convex formulation for the reconstruction of low-rank tensor, and the existing algorithm ADMM. We then apply our hybrid algorithm to ADMM to improve its computation efficiency, and show numerically that the new algorithm performs better than ADMM in all three formulations with similar recovery rates and shorter computation time.

Chapter 4 considers the discriminative methods for predicting running time of servers in large-scale computing systems. We first discuss the map/reduce design of the parallel computing systems, and the associated diagnostic problems in practice. We then study the penalized functional regression, and apply our hybrid algorithm to improve it. We compare it against an additive generalized linear model and logistic regression on four datasets from DyradLINQ of Microsoft. We study the performance of three methods in binary classification and continuous prediction. Our results show that penalized functional regression is uniformly better than the other two methods. We suggest it in predicting running time servers in large-scale computing systems.

Chapter 5 studies the generative methods for diagnostics of large-scale computing systems. We propose three data-driven models, HMRCA, HSMRCA and HMM, to classify the servers in computing systems. HMRCA and HSMRCA are designed to capture the regime-switching behavior we observe from data. We also discuss the es-

timation problems of Monte Carlo simulation in these models, and design a blocked Gibbs sampling algorithm to improve the convergence of simulated Markov chain. We apply our models to the datasets from DyradLINQ of Microsoft, and show that the regime-switching property improves the classification accuracy.

Chapter 6 concludes our contributions in this thesis.

CHAPTER 2

HYBRID PRINCIPAL COMPONENT ANALYSIS IN HIGH DIMENSIONAL LOW-RANK MATRIX

Matrix orthogonality constraints have important influence in many scientific research areas. In particular, minimization with the orthogonality constraints is widely used in polynomial computation, combinatorial mathematics, eigenvalue calculation, sparse principal component analysis, matrix rank specification, etc. These problems are challenging because the constraints are computationally expensive to preserve during iterations.

One of the interesting problems is the following optimization problem with orthogonality constraints,

$$\min_{X \in R^{n \times p}} \mathscr{F}(X) \quad s.t. \quad X^T X = I \tag{2.1}$$

where I is the identity matrix and $\mathscr{F}(X)$ is a differentiable function. This problem is difficult because it is challenging to directly solve the nonlinear and nonconvex constraints. As a result, iterative methods are commonly used instead. An curvilinear search algorithm (Algorithm 1, see below) was previously proposed in Wen and Yin (2013) to solve (2.1), and was proved to be efficient and robust in various test cases. However, the application of Algorithm 1 in high dimensional low-rank matrices were not carefully discussed in Wen and Yin (2013). In Section 2.1, we propose a modified curvilinear algorithm to solve the linear eigenvalue problem for high dimensional low-rank matrices.

Principal component analysis (PCA) is a classical method for data analysis and dimension reduction. However, each principal component is a linear combination of all the original attributes, so it is difficult to interpret the result. Sparse principal component analysis (SPCA) extends standard PCA, and produces more zero loadings in modified

principal components. In Section 2.2, we apply the modified curvilinear algorithm in Section 2.1 to an augmented Lagrangian method (Lu and Zhang 2012).

Algorithm 1: A gradient descent curvilinear algorithm (2.1)

1 Given an initial point X_0 with $X_0^T X_0 = I$;

2 Set $k = 0$, $\varepsilon \geq 0$ and $0 < \rho_1 < \rho_2 < 1$;

3 **while** *true* **do**

4 Generate A according to $A = GX^T - XG^T$;

5 Set $Y_k(\tau_k) = (I + \frac{\tau_k}{2}A)^{-1}(I - \frac{\tau_k}{2}A)X$.

6 Choose a step size τ_k satisfying the Armijo-Wolfe conditions;

$$\mathscr{F}(Y_k(\tau_k)) \leq \mathscr{F}(Y_k(0)) + \rho_1 \tau_k \mathscr{F}_\tau^{'}(Y_k(0))$$

$$\mathscr{F}_\tau^{'}(Y_k(\tau_k)) \geq \rho_2 \mathscr{F}_\tau^{'}(Y_k(0))$$

Update $X_{k+1} = Y_k(\tau_k)$;

7 If $\| \bigtriangledown \mathscr{F}_{k+1} \| \leq \varepsilon$, then stop; Otherwise, $k = k + 1$ and go to step 4;

8 **end**

2.1 Eigenvalue Problems for high dimensional Low-Rank Matrix

Our goal is to calculate the largest few eigenvalues of a high dimensional low-rank symmetric matrix. For any symmetric matrix $\Sigma \in R^{n \times n}$ and unitary matrix $X \in R^{n \times p}$, when the columns of X form an orthogonal basis of the eigenspace, we obtain the maximum of the trace of $X^T \Sigma X$. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of Σ. The sum of the p-largest eigenvalues is then

$$\sum_{i=1}^{p} \lambda_i = \max_{X \in R^{n \times p}} tr(X^T \Sigma X) \quad s.t. \quad X^T X = I. \tag{2.2}$$

Since (2.2) is a special case of (2.1), we apply Algorithm 1 to (2.2) to calculate the sum of the largest p eigenvalues, where we change the computation in Algorithm 1 in two aspects (Section 2.1.1 and Section 2.1.2).

2.1.1 Reducing matrix inversion by Sherman-Morrison-Woodbury

In Algorithm 1, when we compute $Y(\tau)$ by $Y(\tau) = (I + \frac{\tau}{2}A)^{-1}(I - \frac{\tau}{2}A)X$, we invert $(I + \frac{\tau}{2}A)$ to preserve the orthogonality constraints. It is computationally more efficient than SVD. In many applications of high dimensional matrices, p is usually much smaller than $n/2$. From the Sherman-Morrison-Woodbury theorem, we only need to invert a smaller matrix with size $2p \times 2p$.

Since $A = GX^T - XG^T$, we rewrite $A = UV^T$ for $U = [G, X]$ and $V = [X, -G]$. Then $I + \frac{\tau}{2}A = I + \frac{\tau}{2}UV^T$, by applying the SMW formula:

$$(B + \alpha UV^T)^{-1} = B^{-1} - \alpha B^{-1}U(I + \alpha V^T B^{-1}U)^{-1}V^T B^{-1},$$

with $B = I$, we obtain $(I + \frac{\tau}{2}A)^{-1} = I - \frac{\tau}{2}U(I + \frac{\tau}{2}V^T U)^{-1}V^T$. With $I - \frac{\tau}{2}A = I - \frac{\tau}{2}UV^T$, we have

$$\begin{aligned} Y(\tau) &= X - \frac{\tau}{2}U\left((I + \frac{\tau}{2}V^T U)^{-1}(I - \frac{\tau}{2}V^T U) + I\right)V^T X \\ &= X - \tau U(I + \frac{\tau}{2}V^T U)^{-1}V^T X. \end{aligned} \tag{2.3}$$

If $p \ll n$, inverting $I + \frac{\tau}{2}V^T U \in R^{2p \times 2p}$ is numerically cheaper than inverting $I + \frac{\tau}{2}W \in R^{n \times n}$, so we use (2.3) to compute $Y(\tau)$ in Algorithm 1.

2.1.2 Line Search with Barzilai-Borwein steps

The gradient descent algorithm (Algorithm 1) is easy to implement, but the Barzilai-Borwein (BB) step size is well known for accelerating the gradient method. Hence, instead of choosing a step size τ_k to satisfy Armijo-Wolfe conditions in Algorithm 1, we set τ_k to be

$$\tau_{k,1} = \frac{tr((S_{k-1})^T S_{k-1})}{|tr((S_{k-1})^T Y_{k-1})|} \quad \text{or} \quad \tau_{k,2} = \frac{|tr((S_{k-1})^T Y_{k-1})|}{tr((Y_{k-1})^T Y_{k-1})}, \tag{2.4}$$

where $S_{k-1} = X_k - X_{k-1}$ and $Y_{k-1} = \bigtriangledown \mathscr{F}(X_k) - \bigtriangledown \mathscr{F}(X_{k-1})$. We also adopt a nonmonotone line search method proposed by Zhang and Hager (2004) and Dai and Fletcher (2005). That is, we generate new points iteratively in the form $X_{k+1} = Y_k(\tau_k)$, where $\tau_k = \tau_{k,1}\delta^h$ or $\tau_k = \tau_{k,2}\delta^h$, and h is the smallest integer satisfying

$$\mathscr{F}(Y_k(\tau_k)) \leq C_k + \rho_1 \tau_k \mathscr{F}'(Y_k(0)), \tag{2.5}$$

where $C_{k+1} = (\eta Q_k C_k + \mathscr{F}(X_{k+1}))/Q_{k+1}$, $Q_{k+1} = \eta Q_k + 1$ and $Q_0 = 1$.

We now formally present the modified algorithm.

Algorithm 2: A modified curvilinear search with Barzilai-Borwein steps

Given an initial point X_0, set $\tau > 0$, ρ_1, δ, η, $\varepsilon \in (0,1)$, $k = 0$;
while $\| \bigtriangledown \mathscr{F}(X_k) \| > \varepsilon$ **do**
    while $\mathscr{F}(Y_k(\tau)) \geq C_k + \rho_1 \tau \mathscr{F}'(Y_k(0))$ **do**
        $\tau = \delta\tau$;
    end
    $X_{k+1} = Y_k(\tau_k)$, $Q_{k+1} = \eta Q_k + 1$, and $C_{k+1} = (\eta Q_k C_k + \mathscr{F}(X_{k+1}))/Q_{k+1}$;
    Set $\tau = \max(\min(\tau_{k+1,1}, \tau_M))$ and $k = k+1$;
end

2.1.3 Numerical Results in Linear Eigenvalue Problem

In this section, we illustrate the computational advantage of Algorithm 2 in the linear eigenvalue problem (2.2). We compare Algorithm 2 against the Matlab function "eigs" in a number of test matrices. We first implement the algorithm in a few randomly generated dense matrices Σ. In Table 2.1, n varies from 5000 to 15000, and $p = 6$ (we calculate the sum of the 6 largest eigenvalues). In this table, "obj" denotes the optimal value of the objective function, "cpu" denotes the CPU time, "err" denotes the relative error between the results from eigs and modified algorithm (Algorithm 2). We see that the two algorithms have similar performance when n is small, but when $n > 10000$, modified curvilinear algorithm (Algorithm 2) is significantly faster than eigs. In Table 2.2, we fix $n = 10000$ and vary p. We see that Algorithm 2 is faster, especially when p is relatively small ($p = 1$).

Table 2.1: Comparision of eigenvalue calculation on random matrices with p = 6

n	5000	8000	10000	12000	15000
eigs					
obj	1.186795e+005	1.905116e+005	2.380515e+005	2.858445e+005	3.580073e+005
cpu	4.795257	11.296551	27.157019	43.195600	145.330882
modified curvilinear					
obj	1.186792e+005	1.905115e+005	2.380477e+005	2.858390e+005	3.580069e+005
cpu	4.078610	7.327771	28.586046	18.588451	53.603562
err	2.52e-006	3.50e-007	1.58e-005	1.93e-005	1.28e-006

In our second experiment, we apply both methods to 5 large sparse matrices ($n \geq$ 80000) from the UFL Sparse Matrix Collection (Davis and Hu 2011). We compute the 6 largest eigenvalues and present the results in Table 2.3. In this table, "nonzeros" denotes the number of non-zero entries in the sparse matrices. We see that Algorithm 2 is competitive in most problems, and significantly faster than "eigs" on problems such

Table 2.2: Comparision of eigenvalue calculation on random matrices with n = 10000

p	1	3	5	7	9
eigs					
obj	3.984335e+004	1.193378e+005	1.985642e+005	2.774991e+005	3.562744e+005
cpu	14.305161	28.298328	23.106653	27.356422	27.338908
modified curvilinear					
obj	3.984330e+004	1.193376e+005	1.985640e+005	2.774922e+005	3.562709e+005
cpu	5.807126	19.767615	12.661969	16.551770	19.016399
err	1.10e-006	1.37e-006	1.10e-006	2.47e-005	9.82e-006

as "$Ga10As10H30$" and "$Ga41As41H72$". However, "eigs" has excellent performance on "*boyd*2" (less than 2 seconds), and is dramatically faster than Algorithm 2. When a matrix has special structures, "eigs" is able to capture them to reduce the computation time.

Table 2.3: Comparision of eigenvalue calculation on real sparse matrices from Sparse Matrix Collection - CISE.UFL

name	*ncvxqp*7	$Ga10As10H30$	*CO*	$Ga41As41H72$	*boyd*2
n	87500	113081	221119	268096	466316
nonzeros	574962	6115633	7666057	18488476	1500397
eigs					
obj	2.368702e+05	7.804780e+03	7.933764e+02	7.805603e+03	1.186079e+10
cpu	16.098919	16.016042	36.971262	33.198001	1.847562
modified curvilinear					
obj	2.368496e+05	7.804515e+03	7.933653e+02	7.805080e+03	1.186050e+10
cpu	14.952167	4.752456	18.869110	9.605550	4.736383
err	8.68e-05	3.40e-05	1.40e-05	6.69e-05	2.46e-05

2.2 Algorithm for Sparse Principal Component Analysis

2.2.1 Hybrid Principal Component Analysis (HPCA)

In the previous section, we have shown that Algorithm 2 is an efficient method in solving the linear eigenvalue problem (2.2), especially for high dimensional matrices. Nonetheless, it only calculates the eigenvalues, but not the eigenvectors. However, it can improve the efficiency of an existing method for sparse principal component analysis.

Sparse principal component analysis (SPCA) has been extensively studied for the last few decades. In this work, we formulate the problem as follows:

$$\begin{aligned} &\max_{X \in R^{n \times p}} tr(X^T \Sigma X) - \rho |X| \\ &s.t. \quad |X_i^T \Sigma X_j| \leq \Delta_{i,j}, \\ &\qquad X^T X = I, \end{aligned} \tag{2.6}$$

where $\Delta_{i,j} \geq 0 (i \neq j)$ are tuning parameters for the correlation of X, and $\rho > 0$ is the penalized parameter of sparsity. This sparse PCA formulation maintains the following three properties of the standard PCA: (1) maximal total explained variance, (2) uncorrelation of principal components, (3) orthogonality of loading vectors.

As a recent work, Lu and Zhang (2012) give an augmented Lagrangian method (Algorithm 3) for solving a generalization of (2.6). The problem is written as

$$\begin{aligned} &\min_{X \in R^{n \times p}} f(X) + P(X) \\ &s.t. \quad g_i(X) \leq 0, \quad i = 1, \ldots, m, \\ &\qquad h_j(X) = 0, \quad j = 1, \ldots, p, \end{aligned} \tag{2.7}$$

where f, g_i and h_j are continuously differentiable, and P is convex but not necessarily

smooth.

Algorithm 3: Augmented Lagrangian method for (2.7)

Set $k = 0$, λ^0, μ^0, $\rho_0 > 0$, $\tau > 0$, $\sigma > 1$;

Find an initial point X^0_{init} and a constant $\Upsilon > \max\{f(X^{feas}), L_{\rho_0}(X^0_{init}, \lambda^0, \mu^0)\}$;

while *true* **do**

Find a candidate solution X^k for the subproblem

$$\min_{X^k} L_{\rho_k}(X^k_{init}, \lambda^k, \mu^k) := w(X^k) + P(X^k)$$

where $w(X^k) =$

$f(X^k) + (\|[\lambda^k + \rho_k g(X^k)]^+\|^2 - \|\lambda\|^2)/(2\rho_k) + \mu^k h(X^k) + \rho_k \|h(X^k)\|^2/2$;

Update Lagrange multipliers

$$\lambda^{k+1} = [\lambda^k + \rho_k g(X^k)]^+, \quad \mu^{k+1} = \mu^k + \rho_k h(X^k)$$

Set $\rho_{k+1} = \max\{\sigma\rho_k, \|\lambda^{k+1}\|^{1+\tau}, \|\mu^{k+1}\|^{1+\tau}\}$;

If $\max_{i \neq j}[|X_i^T \Sigma X_j| - \Delta_{i,j}]^+ \leq \varepsilon_I$, $\max|X^T X - I| \leq \varepsilon_E$, and $|L_\rho(X, \lambda, \mu) - f(X)|/\max(|f(X)|, 1) \leq \varepsilon_O$ then stop; Otherwise, $k = k + 1$ and continue;

end

Notice that (2.6) is a special case of (2.7). In Lu and Zhang (2012), they directly apply Algorithm 3 , which they called "Alspca", to (2.6). They have shown that it properly controls the orthogonality and correlation of the components X while maintaining the sparsity. However, Algorithm 3 uses the standard SVD decomposition to find an initial point, which is not computationally efficient, and sometimes even intractable in high dimensional matrices. Thus, we incorporate Algorithm 2 into Algorithm 3 to solve (2.6) for high dimensional matrices. That is, we use Algorithm 2 to get a feasible starting

point X_{init}^0 for Algorithm 3 in our high dimensional matrix computation. Our final hybrid algorithm is listed as follows:

Algorithm 4: A hybrid algorithm for high dimension matrix ("HPCA")

1 Use Algorithm 2 to find an plausible initial point for X_{init}^0;

2 Use X_{init}^0 in Algorithm 3 to find the optimal point X.

The idea of HPCA is to quickly find the orthogonal components X using Algorithm 2 as an initial point, and then use Algorithm 3 to reduce the correlations of X, increase the sparsity of X, while maintaining their orthogonality.

2.2.2 Numerical Results of HPCA in Sparce Principal Component Analysis

In this section, we explore the numerical performance of HPCA (Algorithm 4) in sparse PCA (2.6) using randomly generated matrices. In particular, we compare HPCA against Alspca and a most commonly used SPCA method, the generalized power methods (GPower) (Journée et al. 2010), w.r.t. total explained variance, correlation of PCs, orthogonality of loading vectors, and computation times. We use two types of generalized power methods, single-unit SPCA via l_1 penality ("$GPower_{l_1}$") and single-unit SPCA via l_0 penality ("$GPower_{l_0}$"). As mentioned by Lu and Zhang (2012), $tr(X^T \Sigma X)$ in (2.6) basically equals the total explained variance of the first p PCs. However, the PCs found by SPCA are not perfectly uncorrelated, and $tr(X^T \Sigma X)$ can overestimate the total explained variance by the PCs due to the overlaps of the individual variances. As a result, Lu and Zhang (2012) introduced the adjusted total explained variance and the

cumulative percentage of adjusted variance (CPAV) for sparse PCs:

$$AdjVar = tr(X^T \Sigma X) - \sqrt{\sum_{i \neq j} (X_i^T \Sigma X_j)^2}, \qquad CPAV = AdjVar/tr(\Sigma), \tag{2.8}$$

In our experiments, we use CPAV as a metric for measuring the total explained variance. In the first one, we try to find the first 6 sparse PCs with the average percentage of zero loadings to be approximately around 80% (80% sparsity). To achieve this, the tuning parameter ρ for Problem (2.6) and the parameters for the GPower methods are chosen properly. The test set includes 1000 full-rank random matrices. The results in Table 2.4 correspond to the matrix with size 200×200 and 1000×1000 separately. In this table, "sparsity" measures the number of zero loadings averaged over all instances. The third row "Ortho(Mean)" gives the average amount of orthogonality of the loading vectors, which is measured by the maximum absolute angles formed by all pairs of loading vectors. Larger values in this row imply better orthogonality. The fourth row "Ortho(Std)" gives the corresponding standard deviation. The average maximum correlation between all pairs of loading vectors is given in row five ("Corr(Mean)"), and the corresponding standard deviation is given in row six ("Corr(Std)"). The rows seven and eight ("CPAV (Mean)", "CPAV (Std)") give the average CPAV (defined in (2.8)) and its standard deviation. The average cpu time and the corresponding standard deviation are given in the last two rows.

From Table 2.4, we see that HPCA and Alspca give nearly identical results, while HPCA is more computationally efficient. Specifically, HPCA spends roughly 20% less computing time in the 200×200 matrix test, and 25% less time in the 1000×1000 matrix test. HPCA is also more stable than Alspca in computation time with a smaller standard deviation in both size of matrices. Also, Alspca and HPCA obtain almost uncorrelated sparse PCs and nearly orthogonal loading vectors, which outperforms the GPower methods. Alspca and HPCA are also more stable in controlling the correlation and orthogonality, as both methods have smaller standard deviations than GPower. In terms of

Table 2.4: Comparision of SPCA methods on randomly generated full-rank matrices with $p = 6$

Method	200×200				1000×1000			
	$GPower_{l_1}$	$GPower_{l_0}$	$Alspca$	$HPCA$	$GPower_{l_1}$	$GPower_{l_0}$	$Alspca$	$HPCA$
Sparsity	0.8008	0.8035	0.8099	0.8007	0.8039	0.8066	0.8091	0.8099
Ortho(Mean)	87.134	87.327	89.991	89.987	86.433	86.730	90.000	89.997
Ortho (Std)	0.533	0.582	0.022	0.030	0.404	0.386	0.022	0.023
Corr (Mean)	0.062	0.066	0.026	0.025	0.049	0.043	0.015	0.014
Corr (Std)	0.018	0.018	0.004	0.004	0.010	0.009	0.002	0.002
CPAV%(Mean)	4.972	4.881	4.982	4.982	1.759	1.923	1.950	1.953
CPAV%(Std)	0.007	0.008	0.010	0.010	0.022	0.019	0.011	0.011
CPU (Mean)	0.063	0.054	5.166	3.915	0.655	0.514	53.623	38.915
CPU (Std)	0.012	0.008	0.518	0.396	0.096	0.083	5.125	3.696

CPAV, there are no significant differences among these three methods. Gpower method, however, is extremely efficient in computation time. This is mainly because Gpower solves two unconstrained differentiable maximization problems, instead of (2.6), for l_1 penality and l_0 penality respectively,

$$\max_{x \in R^n} \sqrt{x^T \Sigma x} - \gamma \|x\|_1, \qquad \max_{x \in R^n} x^T \Sigma x - \gamma \|x\|_0, \tag{2.9}$$

where $\gamma > 0$ is the sparsity controlling parameter. GPower does not have any constraint on the correlation and orthogonality of PCs, and the resulting variances of the first few PCs are not ordered either.

In the second experiment, we randomly generate 1000 200×200 and 1000 1000×1000 matrices with rank 10 and fixed eigenvalues 100, 89.2, 78.4, 67.7, 56.9, 46.1, 35.3, 24.6, 13.8, 3.0. We set $p = 6$ and 80% sparsity, and present the results in Table 2.5. We see that GPower has substantially higher maximum correlation than both Alspca and HPCA. This is because Gpower does not have constraints on the correlation of loading vectors, and is hence sensitive to the rank of the test matrices. GPower also has smaller

Table 2.5: Comparision of SPCA methods on randomly generated low-rank matrices with $p = 6$

Method	200×200				1000×1000			
	$GPower_{l_1}$	$GPower_{l_0}$	$Alspca$	$HPCA$	$GPower_{l_1}$	$GPower_{l_0}$	$Alspca$	$HPCA$
Sparsity	0.8085	0.8082	0.8073	0.8073	0.8070	0.8097	0.8025	0.8010
Ortho(Mean)	85.098	85.262	89.968	89.955	87.865	87.839	89.945	89.959
Ortho (Std)	1.438	1.242	0.054	0.065	0.686	0.784	0.052	0.053
Corr(Mean)	0.637	0.684	0.159	0.158	0.710	0.750	0.026	0.025
Corr (Std)	0.095	0.099	0.047	0.043	0.071	0.082	0.002	0.002
CPAV%(Mean)	40.517	40.821	42.891	42.892	32.181	33.303	49.303	49.303
CPAV%(Std)	0.281	0.302	0.118	0.114	0.215	0.194	0.119	0.118
CPU (Mean)	0.008	0.007	3.493	2.512	0.197	0.170	74.294	58.961
CPU (Std)	0.004	0.004	0.411	0.238	0.030	0.022	8.125	5.996

CPAV and orthogonality than Alspca and HPCA. When we increase the matrix size from 200×200 to 1000×1000, GPower has a smaller the average CPAV (from 40% to 33%) while Alspca and HPCA have a higher one (from 43% to 49%). In terms of computation time, GPower is more efficient than Alspca and HPCA, which we have discussed in the previous experiment. Combining Table 2.4 with Table 2.5, we find that Alspca and HPCA spend more time to deal with low rank matrices than with full rank ones, while the opposite is true for GPower.

2.3 Application of HPCA in Equity Statistical Arbitrage

We now apply HPCA (Algorithm 4) to a statistical arbitrage strategy where we trade a portfolio of stocks against market factors, which are defined using the standard PCA, or the corresponding sector exchange-traded funds (ETFs). We extend the work of Avellaneda and Lee (2010) to define market factors using HPCA, and examine their perfor-

mance from 2007 to 2013.

2.3.1 Problem Formulation

In statistical arbitrage, we decompose the stock return into two parts: the return that can be explained by some common market factors, and the idiosyncratic return. The idiosyncratic part can be modeled statistically by stationary mean reversion process. This paradigm is based on the assumption of market overreaction, for instance, some stocks are temporarily mispriced with respect to other reference stocks or indices, and open market operations will revert their prices back in a short period of time (Lo and MacKinlay 1990). The trading strategy for this paradigm is called "pairs-trading" (Avellaneda and Lee 2010). Suppose that stock S and reference security Q in the same industry have similar characteristics, and we expect their returns are tractable with each other. The general decomposition of stock return based on market factors is then

$$\frac{dS_t}{S_t} = \beta_0 dt + \beta_1 \frac{dQ_t}{Q_t} + dX_t, \tag{2.10}$$

where S_t is the time-series price of the stock, Q_t is the time-series price of the reference security respectively, X_t is a stationary mean reverted process, β_0 is the time drift, and β_1 is the regression coefficient. In (2.10), X_t is the idiosyncratic part. If β_1 is chosen carefully, the mean reverted property of X_t guarantees that the long-short portfolio of S and Q oscillates near its statistical equilibrium. However, it is extremely difficult to find such a pair S and Q in practice. Instead, people usually employ an extension of (2.10), called "generalized pairs-trading",

$$\frac{dS_t}{S_t} = \beta_0 dt + \sum_{j=1}^{n} \beta_j F_t^{(j)} + dX_t, \tag{2.11}$$

where market factors $F_t^{(j)}$, $j = 1, \ldots, n$ are defined by other reference securities.

2.3.2 Methods for Modeling Market Factors

The first method for finding market factors in (2.11) is to choose corresponding sector ETFs. ETFs are investment assets traded on stock exchanges that provide exposure to a certain sector. Since the late 1990s, more and more investors choose ETFs to diversify their portfolio because of the following reasons: (1) ETFs provide straightforward tradable access to an individual sector or industry, (2) nowadays there are thousands of ETFs available in the worldwide markets, (3) ETFs are very liquid.

From the perspective of statistical arbitrage, it is advantageous to choose ETFs as market factors in (2.11). The first reason is that ETFs allow a stock to be traded directly against corresponding sector ETFs when the stock price diverges from the statistical equilibrium, hence it is straightforward to incorporate ETFs into the model. Secondly, it is intuitive to interpret the factor loadings when we choose ETFs as market factors. Nevertheless, selecting ETFs requires some prior knowledge of the economy and industry, and most ETFs have the priori capitalization bias, that is, ETFs holdings give more weight to large capitalization companies. Also, ETFs are correlated, which brings some redundancies into the model. These issues push people towards some other approaches.

As a fundamental tool of data analysis, principal component analysis (PCA) can also be applied to the correlation matrix of the stock returns to extract market factors in (2.11). Avellaneda and Lee (2010) justified that we can consider principal components as long-short portfolio of industry sectors. We now describe the procedure of deriving the market factors. Assume that $\{S_{i,t}\}$, $i = 1,\ldots,N$, $t = 0,\ldots,M$ are adjusted closing stock prices for N stocks over the past M time periods. The stock price return and the standardized price return are

$$R_{i,t} = \frac{S_{i,t} - S_{i,t-1}}{S_{i,t-1}} \quad \text{and} \quad Y_{i,t} = \frac{R_{i,t} - \bar{R}_i}{\bar{\sigma}_i}, \tag{2.12}$$

where

$$\bar{R}_i = \frac{1}{M}\sum_{t=1}^{M} R_{i,t} \quad \text{and} \quad \bar{\sigma}_i^2 = \frac{1}{M-1}\sum_{t=1}^{M}(R_{i,t} - \bar{R}_i)^2. \tag{2.13}$$

We then generate the $N \times N$ correlation matrix of empirical returns with the $(i,j)^{th}$ entry to be

$$\rho_{i,j} = \frac{1}{M-1}\sum_{t=1}^{M} Y_{i,t} Y_{j,t}. \tag{2.14}$$

PCA then extracts the eigenvalues and eigenvectors of the correlation matrix (2.14), denoted by λ_j and $v^{(j)} = (v_1^{(j)}, \ldots, v_N^{(j)})$, $j = 1, \ldots, N$. Without loss of generality, we assume $\{\lambda_j\}_{j=1}^N$ are in a decreasing order, and the eigenvectors $\{v^{(j)}\}_{j=1}^N$ of the correlation matrix are tied with the market factors in the following way: the entries of an eigenvector corresponds to the weights in a particular stock. If we scale the weights by the volatility of the stock, we can view each eigenvector as a portfolio that holds weights in each stock (Avellaneda and Lee 2010). The weights are

$$Q_i^{(j)} = \frac{v_i^{(j)}}{\bar{\sigma}_i}. \tag{2.15}$$

Incorporating the stock return, we get the eigenportfolio return series or market factor return series,

$$F_t^{(j)} = \sum_{i=1}^{N} \frac{v_i^{(j)}}{\bar{\sigma}_i} R_{i,t}. \tag{2.16}$$

From (2.16) we conclude that each stock return can be decomposed into the projection on market factors and idiosyncratic residuals, as in (2.11).

PCA delivers a set of orthogonal market factors. Compared with the ETFs approach, it does not require any specific prior information about the economy. However, PCA has its own drawback – since each eigenportfolio is a linear combination of all stock returns, and the loadings are usually nonzero, it is often difficult to interpret the derived eigenportfolios. Several previous authors have dealt with this issue. For instance, Laloux

et al. (2000) linked the first eigenportfolio, which corresponds to the eigenvector with the largest respective eigenvalue, with the market, or equivalently, a generalized index on the market. Avellaneda and Lee (2010) noted an interesting phenomenon: when we rank the coefficients of the eigenvectors in a decreasing order, different company stocks presented by adjacent coefficients tend to be in the same industry. But this is not necessary true for some noisy eigenvectors.

To improve the interpretability of PCA, we use SPCA as an alternative. The procedure for deriving the market factors is exactly the same as with PCA. The only difference is to use SPCA instead of PCA to extract eigenvalues and eigenvectors from correlation matrix (2.14). In our implementation of SPCA, we use HPCA (Algorithm 4) defined in Section 2.2.1.

Both PCA and SPCA reduce the dimension of original dataset by projecting along the orthogonal eigenvectors, which represent the maximum variance of the data in each direction. We choose the number of eigenvectors using two criteria: (1) A fixed number of eigenvectors, in which case the total variance explained changes every time, (2) flexible number of eigenvectors, in order to keep a specific total variance explained every time. We apply both criteria in Section 2.3.4

2.3.3 Trading Signals and Arbitrage Strategy

In the previous section, we have discussed three methods (ETFs, PCA, SPCA) to find market factors $\{F_t^{(j)}\}$ in (2.11). Now we follow the approach in Avellaneda and Lee (2010) to generate trading signals, which can be used for trading against any market factors.

We first compute the regression model for each stock i, $i = 1, \ldots, N$, using a window of M days,

$$R_{i,t} = \beta_{i,0} + \sum_{j=1}^{n} \beta_{i,j} F_{i,t}^{(j)} + \varepsilon_{i,t}, \quad t = 1, \ldots, M, \tag{2.17}$$

where $R_{i,t}$ is the return of stock i at time t, $\beta_{i,0}$ is the drift, $\{F_t^{(j)}\}_{j=1}^{n}$ and $\{\beta_{i,j}\}_{j=1}^{n}$ are its market factors and the corresponding regression coefficients for stock i, and $\varepsilon_{i,t}$ is the residual of stock i at time t.

We then generate the cumulative residual returns, which is a discrete time proxy for the residual time-series process. We assume that this time series follows an $AR(1)$ process,

$$X_{i,t} = \sum_{k=1}^{t} \varepsilon_{i,k}, \quad t = 1, \ldots, M. \tag{2.18}$$

$$X_{i,t+1} = a_i + b_i X_{i,t} + \zeta_{i,t+1}. \quad t = 1, \ldots, M-1. \tag{2.19}$$

For each stock we then compute an "s-score" (Avellaneda and Lee 2010), modeling the distance of the residual returns from the statistical equilibrium. Based on this score, we identify when the stock is away from the equilibrium, and enter the mean reversion position to restore the equilibrium. The s-score for stock i at time t is defined by

$$s_{i,t} = \frac{X_{i,t} - m_i}{\sigma_{i,t}}, \tag{2.20}$$

where

$$m_i = \frac{a_i}{1 - b_i} \quad \text{and} \quad \sigma_{i,t} = \sqrt{\frac{Var(\zeta_{i,t})}{1 - b_i^2}}. \tag{2.21}$$

We then open or close trades as follows:

- long to open the position if $s_{i,t} < -1.25$,
- short to open the position if $s_{i,t} > 1.25$,

- close the short position if $s_{i,t} < 0.75$,
- close the long position if $s_{i,t} > -0.5$.

The cutoff values 1.25, 0.75, and 0.5 are determined empirically based on the simulated strategies from 2000 to 2004 (Avellaneda and Lee 2010). We use these values consistently across all three methods for modeling market factors.

2.3.4 Backtesting Results

To evaluate the performance of an arbitrage strategy we backtest it using the trading signals from Jan 1, 2007 to Dec 31, 2013. We determine the tradable set of stocks on the starting date d based on the following criteria:

- The stock must be a member of the reference index on date d
- The market capital on date d must be greater than certain threshold, e.g. $\$1bn$

On date d we open new positions only on stocks in the tradable set. We define a rebalance period, say after 60 days, when we recompute the tradable set from the most recent reference index constituents. In between these rebalance periods, the tradable set remains constant. We examine the returns of each approach in Section 2.3.2 on a portfolio of \$100 over the past 6 years, from 2007 to 2013. The portfolio consists of stocks from the constituents of *S&P* 500.

There are two alternative implementations for ETFs approach, (1) synthetic ETFs comprised of 15 capitalization-weighted industry indices (Avellaneda and Lee 2010); (2) actual ETFs in the market. For PCA and SPCA, we set the number of market factors in two ways, (1) fixed, e.g., 5, 10, or 15; (2) flexible, explaining 45%, 55%, or 65% of

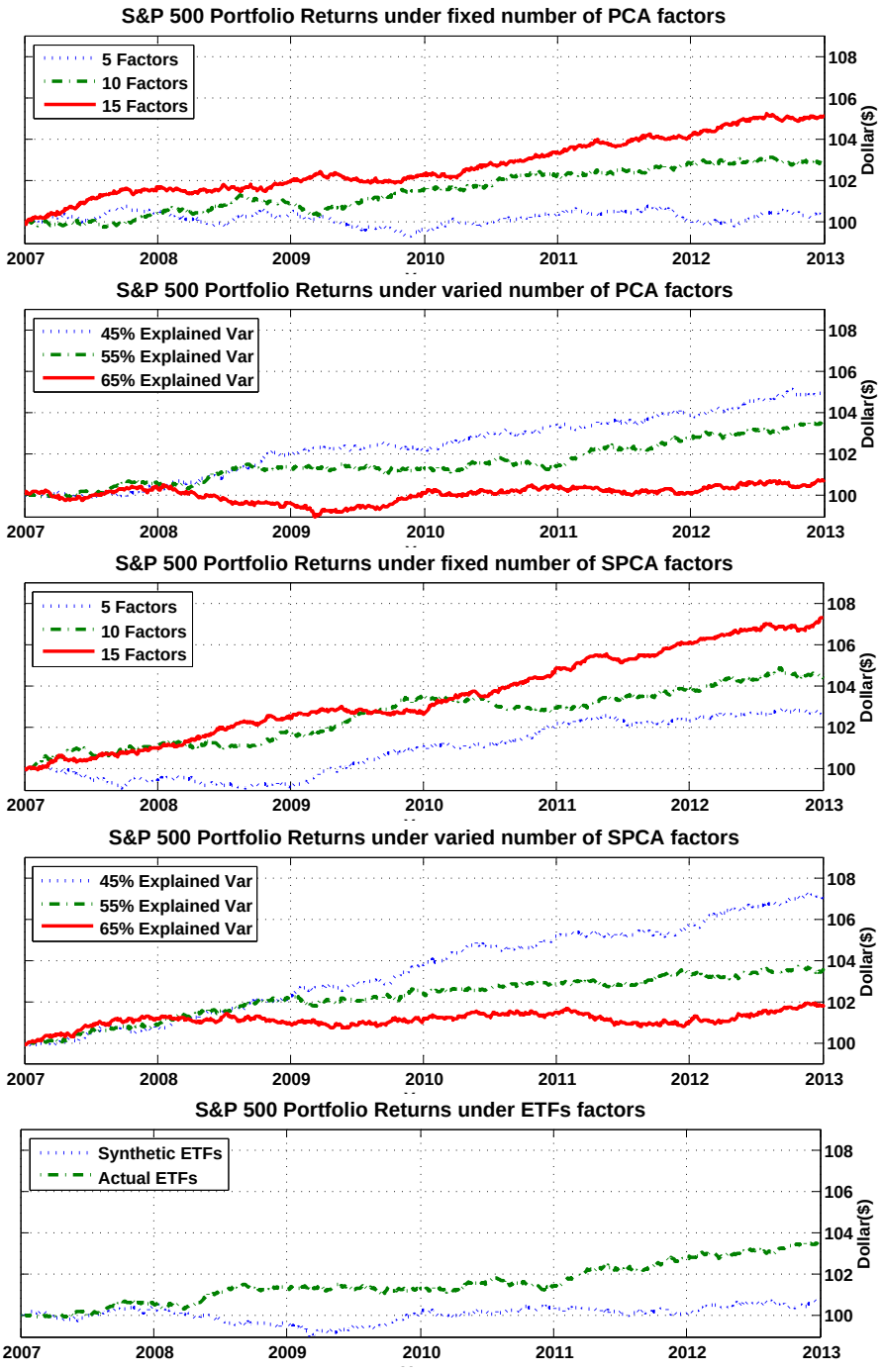

Figure 2.1: Value of a $100 portfolio from *S&P* 500

variance. For simplicity, we name these factoring approaches by "n-factor PCA/SPCA" ($n = 5, 10, 15$), and "p-var PCA/SPCA" ($p = 45\%, 55\%, 65\%$) accordingly.

The backtesting results are displayed in Figure 2.1. These plots show the change of portfolio values while considering transaction costs – 10 basis points per trade. The first plot comes from PCA with a fixed number of factors. Here 5-factor PCA performs the worst, with almost no value increase after 6 years. 15-factor PCA provides the highest return (over 5%). This is partially because 5 market factors are not able to capture sufficient information. The second plot shows the cumulative returns from PCA with varied number of factors. We find that 45%-var PCA produces the highest return (4%). A possible reason is that 45%-var PCA allows enough variance in the residual process (2.18) to produce effective trading signals.

In Figure 2.1, the third and the fourth plots are returns from SPCA with fixed and varied number of factors respectively. From these plots, we see that SPCA generates significantly better results than PCA. It raises returns by almost 2%. We interpret its advantages as follows: the eigenportfolios from SPCA have more zero loadings, so we have fewer trades for each eigenportfolio. As a result, the transaction costs are reduced consistently. Also, the trading strategy from SPCA is faster to implement than the one from PCA. The last plot shows the returns from ETFs market factors. It is apparent that actual ETFs are better than synthetic ETFs. One possible argument is that actual ETFs are traded, so they provide better market price information. However, both methods are not comparable with 15-factor model from PCA or SPCA.

We also compare the Sharpe Ratios of the factoring approaches. Sharpe Ratio is a risk–adjusted measure proposed by Sharpe (1998). To calculate the Sharpe Ratio, we subtract the return of the reference security from the expected return of the portfolio,

and then divide it by the volatility of the portfolio return,

$$S = \frac{E(R_p) - R_r}{\sigma_p}, \tag{2.22}$$

where R_p and R_r are returns from the portfolio and the reference security, and σ_p is the volatility of the portfolio.

Table 2.6: Comparision of factoring approaches on a \$100 portfolio from *S&P* 500

Factoring Approach	Sharpe Ratio	
	PCA	SPCA
5-factor	0.0217832	0.1398662
10-factor	0.1483421	0.2483721
15-factor	0.2453149	0.3517823
45%-var	0.2110375	0.3408152
55%-var	0.1031275	0.1129604
65%-var	0.0418703	0.0939142

Table 2.6 lists the Sharpe Ratios of arbitrage portfolios from 2007 to 2013. We present results for the six factoring approaches. Table 2.6 shows that SPCA improves the Sharpe Ratios by nearly 0.1%, except for 55%-var and 65%-var cases. The reason why PCA performs uniformly worse than SPCA is because there are fewer transaction costs for eigenportfolios generated from SPCA. Thus, we recommend SPCA for modeling market factors in statistical arbitrage.

2.4 Conclusion

In this chapter, we change the curvilinear search algorithm in Wen and Yin (2013) by reducing matrix inversion and accelerating line search. We incorporate it with Alspca

in Lu and Zhang (2012) to get HPCA (Algorithm 4). HPCA is an algorithm for finding sparse and near-uncorrelated principal components, with orthogonal loading vectors while maintaining as much of the total explained variance as possible. It is computationally more efficient than Alspca on randomly generated problem. As a method that inherits the advantages of curvilinear algorithm and Alspca, it is most desirable for SPCA.

We also apply SPCA to the equity statistical arbitrage problem (Avellaneda and Lee 2010) by implementing HPCA algorithm. Our backtesting results show that SPCA improves PCA by providing more interpretable eigenportfolios, and reducing transaction costs. Moreover, the strategies from SPCA are more executable since we trade fewer stocks for each eigenportfolio. In conclusion, we recommend SPCA for statistical arbitrage.

CHAPTER 3

HYBRID PRINCIPAL COMPONENT ANALYSIS IN LOW-RANK TENSOR ESTIMATION

Data analysis of multi-way arrays or tensors have drawn more and more attention in both academia and industry recently. Over the last few years, one special case of tensor, the application of convex optimization in low-rank matrix estimation, has been intensively investigated in the following pioneering literature Fazel et al. (2001), Srebro et al. (2004), Evgeniou and Pontil (2007), Tomioka and Aihara (2007). Also, the concept of trace norm (a convex function) has been introduced to replace the matrix rank (a non-convex function) in convex estimation of low-rank matrix reconstructions. Tomioka et al. (2010) extends the trace norm regularization from matrices to tensors, and proposes three formulations for estimating of low-rank tensors. In this work, we apply HPCA method (Algorithm 4) to alternating direction method of multipliers (ADMM) (Gabay and Mercier 1976), and show numerically that the modified method solves the optimization problems under the three extended formulations and is computationally more efficient and stable.

3.1 Matrix and Tensor

3.1.1 Rank and Trace Norm

The rank r of a $m \times n$ matrix X is the maximum number of linearly independent row/column vectors of X, or the number of nonzero eigenvalues or singular values of X. In the singular value decomposition (SVD) of $X = U\Sigma V^T$, $U \in R^{m\times r}$ and $V \in R^{n\times r}$ are matrices with orthogonal column vectors, and $\Sigma \in R^{r\times r}$ is a diagonal matrix with

diagonal elements $\{\sigma_i\}_{i=1}^{r}$ to be the singular values of X. The trace norm is defined to be trace of Σ, i.e. $\|X\|_{trace} = \sum_{i=1}^{r} \sigma_i$. In contrast to matrix rank, which is a nonconvex function, the trace norm is a convex function – the tightest convex lower bound of the matrix rank (Recht et al. 2010). The trace norm is widely used as a penalization term, and its non-differentiability makes many singular values of X zero when $\|X\|_{trace}$ is included in the objective function. Cai et al. (2010) proposes the following minimization problem

$$\min_{X} \frac{1}{2}\|X - D\|_{Fro}^{2} + \lambda \|X\|_{trace}, \tag{3.1}$$

where $\|\cdot\|_{Fro}$ is the Frobenius norm, D is the original input matrix, and λ is a penalization constant. The analytic solution of (3.1) is called spectral soft-thresholding operator by Cai et al. (2010),

$$prox_{\lambda}^{tr}(D) = U\, max(\Sigma - \lambda, 0)\, V^{T}, \tag{3.2}$$

where $D = U\Sigma V^{T}$ is the SVD of the original input matrix D. Since the maximum operation sets the singular values of D to be zero if they are less than λ, X is usually a low-rank matrix (Tomioka et al. 2010).

3.1.2 Tensor Representation and Rank

For multiple-way arrays or tensors, there are several extended definitions of matrix rank, and the corresponding decompositions. One of the most popular decompositions is the Tucker decomposition. In Tucker decomposition, a tensor $\mathscr{X} \in R^{n_1 \times n_2 \times \cdots \times n_k}$ can be represented by k different unfoldings of mode-i fibers, denoted by $X_{(i)}$, $i = 1, \ldots, k$, where k is the order (number of dimensions) of the tensor $\mathscr{X}$. Fibers are the analogue of matrix columns in tensors. In an order k tensor, there are k different mode-i fibers, $i = 1, \ldots, k$. For instance, in an order 2 tensor (i.e. a matrix), the column is a mode-1

fiber and the row is a mode-2 fiber. As a result, the mode-i unfolding $X_{(i)}$ is a $n_i \times n_{-i}$ matrix by concatenating the mode-i fibers, where $n_{-i} = \prod_{j \neq i} n_j$ (Kolda and Bader 2009), (Tomioka et al. 2010). An order k tensor $\mathscr{X}$ has a rank$-(r_1, r_2, \ldots, r_k)$ if $X_{(i)}$ has a rank r_i, $i = 1, \ldots, k$.

In Tucker decomposition, a rank$-(r_1, r_2, \ldots, r_k)$ tensor $\mathscr{X}$ is written as

$$\mathscr{X} = \mathscr{C} \times_1 U_1 \times_2 U_2 \cdots \times_k U_k, \tag{3.3}$$

where $\mathscr{C} \in R^{r_1 \times r_2 \times \cdots \times r_k}$ is the core tensor, and $U_i \in R^{n_i \times r_i}$, $i = 1, \ldots, k$, are the orthogonal matrices from the singular value decomposition of the mode-i unfolding $X_{(i)}$, i.e. $X_{(i)} = U_i \Sigma_i V_i$. A low rank tensor is a tensor that has a low rank matrix unfolding.

3.2 Trace Norm Penalization for Reconstruction of Low-Rank Tensor

In this section, we first introduce three strategies of trace norm penalization for the reconstruction of partially observed low rank tensors in Tomioka et al. (2010), and then talk about the application of HPCA method in these three strategies.

3.2.1 Single Unfolding Penalization

The first strategy in Tomioka et al. (2010) is to minimize the trace norm of a low rank unfolding for a given tensor $\mathscr{X} \in R^{n_1 \times \cdots \times n_k}$. That is, if we assume the mode-i unfolding $X_{(i)}$ of $\mathscr{X}$ is low rank, we can generalize (3.1) to the following problem

$$\min_{\mathscr{X}} \frac{1}{2} \|\mathscr{L}(\mathscr{X}) - y\|^2 + \lambda \|X_{(i)}\|_{trace}, \tag{3.4}$$

where $y \in R^m$ is the observation vector, $\mathscr{L} : R^{n_1 \times \cdots \times n_k} \to R^m$ is an linear operator, $X_{(i)} \in R^{n_i \times n_{-i}}$ is the mode-i unfolding of $\mathscr{X}$, and λ is the penalization constant. (3.4) converts the estimation of a tensor $\mathscr{X}$ into the estimation of a low rank matrix $X_{(i)}$. Candès and Recht (2009) showed that if the rank of $X_{(i)}$ is not too high, $X_{(i)}$ can be estimated perfectly from a small number of samples, and the tensor $\mathscr{X}$ is recovered perfectly. This strategy is straightforward, but its success depends on the choice of the mode to unfold the tensor. For low dimensional tensors, we can apply the above strategy to all the unfoldings of the tensors, and find the best unfolding. However, it is computationally expensive, and can be intractable when the dimensional is large.

(3.4) is a convex optimization problem. We reformulate it in the following way,

$$\min_{x,Z_i} \frac{1}{2}\|\mathscr{L}(x) - y\|^2 + \lambda \|Z_i\|_{trace}, \quad \text{s.t.} \quad P_i x = z_i, \tag{3.5}$$

where $Z_i = X_{(i)} \in R^{n_i \times n_{-i}}$ is just an auxiliary variable, $x \in R^n$ and $z_i \in R^n$ are vectorizations of $\mathscr{X}$ and Z_i respectively ($n = n_1 \times \cdots \times n_k$), and $P_i \in R^{n \times n}$ is the matrix representation of mode-i unfolding with $P_i^T P_i = I_n$.

3.2.2 Multiple Unfolding Penalization

The second strategy in Tomioka et al. (2010) is an extension of (3.4). Instead of minimizing one mode unfolding of a tensor, this strategy minimizes all mode unfoldings of a tensor.

$$\min_{\mathscr{X}} \frac{1}{2}\|\mathscr{L}(\mathscr{X}) - y\|^2 + \sum_{i=1}^{k} \lambda_i \|X_{(i)}\|_{trace}, \tag{3.6}$$

where $X_{(i)} \in R^{n_i \times n_{-i}}$ is the mode-i unfolding of $\mathscr{X}$, and λ_i is the corresponding penalization constant for $X_{(i)}$.

Similar to (3.4), (3.6) is also a convex optimization problem that can be formulated as follows,

$$\min_{x,Z_1,\ldots,Z_k} \frac{1}{2}\|\mathscr{L}(x)-y\|^2+\sum_{i=1}^{k}\lambda_i\|Z_i\|_{trace}, \quad \text{s.t.} \quad P_i x = z_i, \quad i=1,\ldots,k, \tag{3.7}$$

where $Z_i = X_{(i)} \in R^{n_i \times n_{-i}}$, $i = 1,\ldots,k$, are auxiliary matrices of $\mathscr{X}$, and z_i, is the corresponding vectorization of Z_i, $i = 1,\ldots,k$.

3.2.3 Mixture Unfolding Penalization

Multiple unfolding penalization penalizes all modes of the tensor $\mathscr{X}$. Yet it is impractical to restrict every mode unfolding to be jointly low rank. Instead, Tomioka et al. (2010) proposes the third strategy, which predicts a mixture of k tensors instead of original tensor $\mathscr{X}$. It formulates the problem as follows,

$$\min_{Z_1,\ldots,Z_k} \frac{1}{2}\|\mathscr{L}(\sum_{i=1}^{k} P_i^T z_i)-y\|^2+\sum_{i=1}^{k}\lambda_i\|Z_i\|_{trace}, \quad i=1,\ldots,k, \tag{3.8}$$

(3.8) is a generalization of (3.7). When $z_i = \frac{1}{k}P_i x$ for all $i = 1,\ldots,k$, (3.8) becomes (3.7) with penalization constants λ_i/k.

3.3 Optimization for Trace Norm Penalization

In section 3.2, we have introduced three strategies for the recovery of low rank tensor. In this section, we describe an efficient algorithm to solve the problems (3.5), (3.7), and (3.8), based on the alternating direction method of multipliers (ADMM) (Gabay and Mercier 1976) (Boyd et al. 2011). We then improve its performance by combining ADMM with HPCA method.

3.3.1 ADMM

Before we proceed to introduce ADMM, let us first consider the following constrained optimization problem, which is a generalization of (3.5), (3.7), and (3.8):

$$\min_{x\in R^n, z\in R^m} f(x)+g(z), \quad \text{s.t.} \quad Ax=z, \tag{3.9}$$

where $A\in R^{m\times n}$ is the transformation matrix, and f, g are both convex functions. A traditional way of solving (3.9) is using the method of multipliers (MM), which minimizes the augmented Lagrangian (AL) function of (3.9),

$$L_\eta(x,z,\alpha)=f(x)+g(z)+\alpha^T(Ax-z)+\frac{\eta}{2}\|Ax-z\|^2, \tag{3.10}$$

where $\alpha\in R^m$ is the Lagrangian multiplier vector, and η is the penalty parameter. The method of multipliers (MM) generates a sequence of primal variables (x,z) and multipliers α by iteratively minimizing (3.10). The updating procedure is

$$(x^{t+1},z^{t+1})=\underset{x\in R^n, z\in R^m}{argmin}\ L_\eta(x,z,\alpha^t), \tag{3.11}$$

$$\alpha^{t+1}=\alpha^t+\eta(Ax^{t+1}-z^{t+1}). \tag{3.12}$$

Under certain conditions, the updating procedure (3.11), (3.12) provide a converged solution for (3.9). However, the conditions are usually not satisfied, and solving the joint minimization of (3.11) is infeasible.

The alternating direction method of multipliers (ADMM) (Gabay and Mercier 1976) (Boyd et al. 2011), also known as split Bregman iterations (Goldstein and Osher 2009), solves the joint minimization of (3.11) by separating it into two parts

$$x^{t+1}=\underset{x\in R^n}{argmin}\, L_\eta(x,z^t,\alpha^t), \tag{3.13}$$

$$z^{t+1}=\underset{z\in R^m}{argmin}\, L_\eta(x^{t+1},z,\alpha^t), \tag{3.14}$$

$$\alpha^{t+1}=\alpha^t+\eta(Ax^{t+1}-z^{t+1}). \tag{3.15}$$

It can be shown that the above algorithm converges to a solution of the original problem (3.9), and this is true for any positive value of η.

3.3.2 HPCA in ADMM

ADMM solves (3.9), and can be directly applied to (3.5), (3.7), and (3.8). For (3.5), we can implement all in closed forms as follows. The steps for (3.7) and (3.8) are similar. For details, please see Tomioka et al. (2010).

$$x^{t+1} = (L^T y + \lambda\eta P_i^T(z^t - \alpha^t)./(1_{obs} + \lambda\eta 1_n), \tag{3.16}$$

$$Z^{t+1} = prox^{tr}_{1/\eta}(P_i x^{t+1} + \alpha^t), \tag{3.17}$$

$$\alpha^{t+1} = \alpha^t + (P_i x^{t+1} - z^{t+1}), \tag{3.18}$$

where $L \in R^{m\times n}$ is the linear transformation matrix, $1_{obs} \in R^n$ is a vector with one for observed elements and zero otherwise, $1_n \in R^n$ is a vector with one for each element, and $./$ is the element wise division. $prox^{tr}_{1/\eta}$ in (3.17) is the spectral soft-thresholding operator defined in (3.2) (Cai et al. 2010). This operator performs a SVD on $P_i x^{t+1} + \alpha^t \in R^{n_i \times n_{-i}}$, which is computationally expensive, sometimes even intractable when the matrix size is large. To solve this problem, we incorporate HPCA method in (3.17), and propose the following algorithm.

Algorithm 5 incorporates HPCA in Step (5) to accelerate the computation for large scale matrices. The algorithm stops when the relative difference between the primal objective value $p(x,z)$ and the largest dual objective value $\max_{t'=1,\ldots,t} d(\alpha^{t'})$ in the past t iterations is small enough (Tomioka et al. 2010). Algorithm (5) is proposed for the first strategy (3.5). Algorithms for the other two strategies (3.7) and (3.8) are similar.

Algorithm 5: Alternating Direction Method of Multipliers with HPCA

Given an initial point x^0, Z^0 and α^0;

Initialization. Set $t = 0$, $\varepsilon \geq 0$, $\lambda > 0$ and $\eta > 0$;

while *true* **do**

  Update x^{t+1}. Set $x^{t+1} = (L^T y + \lambda\eta P_i^T(z^t - \alpha^t)./(1_{obs} + \lambda\eta 1_n)$;

  Perform HPCA. Apply HPCA to $P_i x^{t+1} + \alpha^t = U\Sigma V^T$;

  Update Z^{t+1}. Set $Z^{t+1} = U\max(\Sigma - 1/\eta)V^T$;

  Update α^{t+1}. Set $\alpha^{t+1} = \alpha^t + (P_i x^{t+1} - z^{t+1})$;

  Stopping check. If $(p(x^{t+1}, z^{t+1}) - \max_{t'=1,\ldots,t} d(\alpha^{t'}))/p(x^{t+1}, z^{t+1}) \leq \varepsilon$, then **STOP**; Otherwise, $t = t + 1$ and continue;

end

3.4 Numerical Results for Reconstruction of Low-Rank Tensor

In this section, we apply the three strategies in Section 3.2 to reconstruct the simulated tensors using ADMM and HPCA-ADMM (Algorithm 5). We generated 100 tensors for each type in Table 3.1. We randomly chose a fraction r of tensor elements for training, and kept the rest for testing ($r = 5\%, 10\%, 15\%, \ldots, 95\%$). For both ADMM and HPCA-ADMM, we calculate the average computation time from the 100 sample tensors, and summarize our results in Figure 3.1 and Figure 3.2.

Figure 3.1 shows the results from tensors with rank $(3,4,5)$. The plots in the first row represent the average computation time when tensors have dimensions $(100,100,40)$; the ones in the second and third rows are from tensors with dimensions $(500,500,200)$ and $(1000,1000,400)$, respectively. For single unfolding penalization (Section 3.2.1), we only present the unfolding with the shortest time. As illustrated in Figure 3.1, the computation time of both methods (ADMM and HPCA-ADMM) decreases as r in-

Table 3.1: Tensors in numerical simulations

Type	Rank	Dimensions
1	$(3,4,5)$	$(100,100,40)$
2	$(3,4,5)$	$(500,500,200)$
3	$(3,4,5)$	$(1000,1000,400)$
5	$(7,8,9)$	$(100,100,40)$
6	$(7,8,9)$	$(500,500,200)$
7	$(7,8,9)$	$(1000,1000,400)$

creases. Except for tensors with dimensions $(100,100,40)$, the computation time drops rapidly before r reaches 20%, and stabilizes afterwords. This means, for low-rank tensors (rank-$(3,4,5)$), once we observe enough elements, the reconstruction time does not take advantage of the extra information we collect. HPCA-ADMM reduces the average computation time by approximately 25% over ADMM across all tested dimensions. This is because HPCA provides higher computational efficiency than PCA. However, ADMM is computationally more stable when the fraction of observed elements is large. This is partially because the performance of HPCA depends on the choice of penalized and controlling parameters in (2.6). A common set of parameters we choose may not work well for some sample tensors.

Figure 3.2 gives the plots for tensors with rank $(7,8,9)$. We see that when r is small (below 30%, 10%, and 5% respectively), all three strategies stop within a short period of time. This is because we do not have enough information, and all strategies fail to reconstruct the tensors. This also explains the first plot in Figure 3.1. As we raise r to 40%, 20% and 10% respectively, the strategies recover more elements of the tensors as they receive more information, hence the computation time rises. Peak time occurs when we have collected the necessary information for tensor reconstruction. As we keep collecting the information till r reaches a "limit point" (50%, 30% and 20% for tensors

with dimensions $(100, 100, 40)$, $(500, 500, 200)$ and $(1000, 1000, 400)$ respectively), the recovery time then decreases. If r exceeds the limit point, the information becomes redundant, and the recovery time stabilizes. By comparing Figure 3.1 and Figure 3.2, we find that the limit points are larger for tensors with higher rank, since higher rank tensors contain more information, and we need more elements to recover them. We also observe that HPCA-ADMM spends 25% less computation time than ADMM, which coinsides with the as the conclusion we draw from Figure 3.1.

3.5 Conclusion

In this chapter, we study the problem of low-rank tensor estimation (Tomioka et al. 2010), and review three existing strategies for solving the problem. We propose the HPCA-ADMM algorithm, which incorporatse HPCA with ADMM to improve the computation efficiency. We apply both ADMM and HPCA-ADMM to simulated large dimensional low-rank tensors. Our numerical results show that both algorithms successfully reconstruct low-rank tensors from their partial observations, while HPCA-ADMM performs nearly 25% faster. Thus, we recommend HPCA-ADMM for low-rank tensor reconstruction.

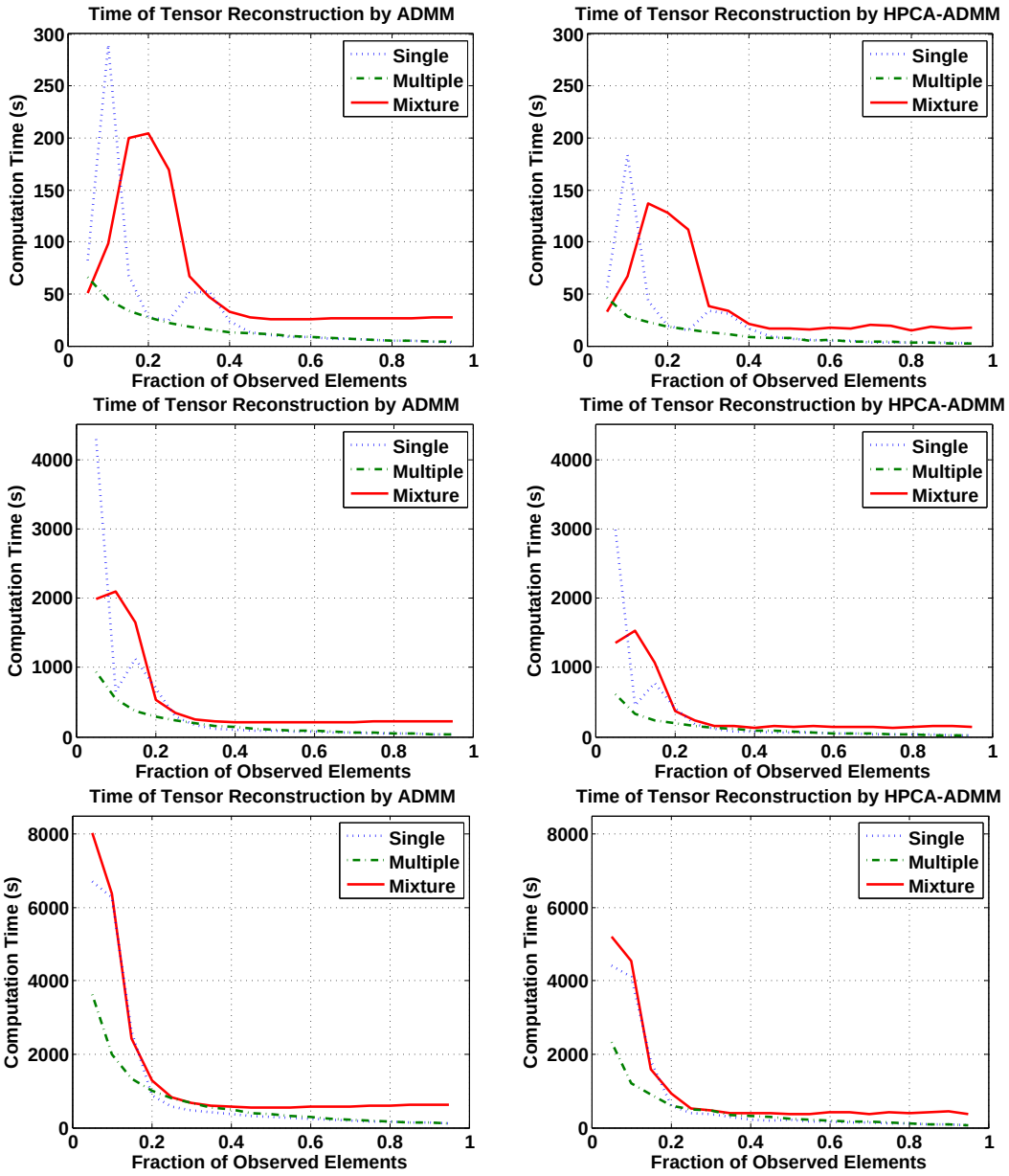

Figure 3.1: Computation of ADMM and HPCA-ADMM in Tensor Reconstruction with rank $(3, 4, 5)$ and dimensions: (1) $(100, 100, 40)$ (above), (2) $(500, 500, 200)$ (middle), $(1000, 1000, 400)$.

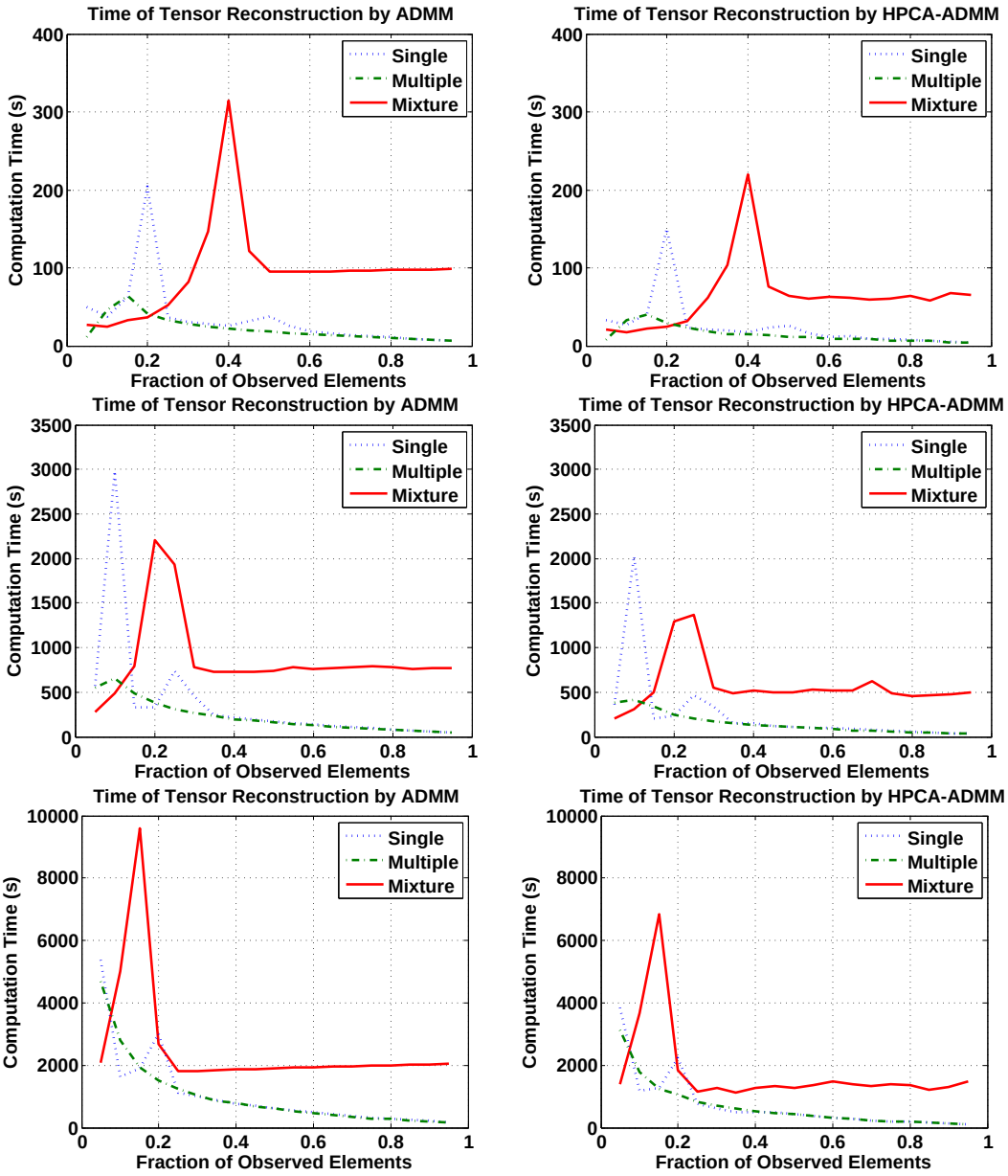

Figure 3.2: Computation of ADMM and HPCA-ADMM in Tensor Reconstruction with rank $(7, 8, 9)$ and dimensions: (1) $(100, 100, 40)$ (above), (2) $(500, 500, 200)$ (middle), $(1000, 1000, 400)$.

CHAPTER 4

DISCRIMINATIVE METHODS FOR PREDICTING PROCESS RUN TIME IN COMPUTING SYSTEMS

Distributed computing platforms following the map/reduce paradigm are the platform of choice for the processing and analysis of the so-called "big data". In these platforms program execution proceeds in stages where multiple servers perform identical tasks simultaneously (albeit on different data). Surprisingly, the run time between servers may vary widely resulting in huge inefficiencies in terms of total computation time, as one server may bottleneck the whole computation of a stage. There may be many causes for this: data skew in the original partition of the data for distribution among the servers, servers competing for data access on the same disk (and also through the network), and (hardware) problems with the servers themselves. Early prediction of abnormally slow performers can be used in system management; for instance one can restart the same task on a different server with local access to the disk (bypassing the need for networking), redistribute the data, or probe the server to verify it is working properly. In this chapter, we discuss the discriminative methods for predicting server run time based on system data early in the process. These methods are based on functional data analysis. We also apply our HPCA algorithm to these methods to improve the computational efficiency. We show that our proposed methods are much more accurate and interpretable than simpler approaches. We validate our methods on real programs running on Microsoft's DryadLINQ platform. In Chapter 5, we study the generative methods based on Hidden Markov/Semi-Markov model.

We introduce methods for fitting models from the performance signals collected on the nodes including CPU utilization, disk utilization, etc., and use this models to identify outliers in the early stages of the computation. Most of our methods are based upon

functional data analysis (Ramsay 2006, Ramsay and Silverman 2002). The most promising method we suggest is an innovative functional linear model proposed by Goldsmith et al. (2010). We have good reason to believe that this approach will be successful as because several nodes execute the same computation we have a significant population to perform this fitting. In addition, as a large portion of these types of jobs are recurrent, we can fit these models off-line and use them on-line in the next execution.

4.1 Estimation Methods

We want to use server data early in the process to predict the run time of the process. The precise run time is of interest, but system operators are also specifically interested in whether the run time will exceed a specified threshold. To address these questions we will fit regression models to the continuous (run time) and binary (run time $>$ threshold) outcomes.

Our predictors are the time series of server metrics, for the first T time periods of the process. Specifically, our prediction of the continuous run time is an estimate of the conditional distribution of the run time, given that the run time is at least T and given the values of the server metrics up to time T. Analogously, our prediction of whether the run time will exceed a threshold d is an estimate of the conditional probability that the run time will exceed d, given that the run time is at least T and given the server metrics up to time T. In practice, our models will be fit for each value of T up to some maximum value above which predictions of run time are no longer useful, or no longer accurate. For instance, in the binary-outcome case $T < d$.

The statistical models are fit using historical data. Then they are applied in real time, as new processes are run. Our models provide the system operators with con-

tinuously updated estimates of the run time of the processes as the processes continue and more data are obtained. The available server metrics commonly include CPU, disk, memory, and network utilization as well as task-specific measurements of system performance (Isard et al. 2007, Yu et al. 2008).

The statistical problem is one of regression with time series predictors. One could solve this problem by simply treating each observation of each predictor as a separate covariate and fitting a standard regression model. However, this does not utilize the time series nature of the predictors. These time series observations can be viewed as discrete, noisy observations of an unknown continuous function. Then we can use regression models designed for functional predictors. As we will see, this leads to more interpretable models and more accurate predictions. This general approach falls into the category of functional data analysis (Ramsay 2006, Ramsay and Silverman 2002).

The most commonly used framework for regression with functional predictors is the generalized functional linear model (1.3). We first focus on the case of a single functional predictor; multiple predictors can be combined easily using an additive model. In the binary classification, we want to predict whether the task completion time of the i^{th} server will be very long $(W_i = 1)$, or regular $(W_i = 0)$, we take g to be the logit link function; while in the prediction of the continuous run time, we take g to be the identity link function. Let $X_i(t)$ be the unknown functional predictor for the i^{th} server. We have noisy observations of $X_i(t)$ at fixed times $t = 1, \ldots, T$; call these observations $\{x_{i,t}\}_{t=1}^{T}$. Let W_i be the outcome variable for the i^{th} server; W_i is either binary or continuous in our context depending on the problem we consider.

4.1.1 Penalized Functional Regression

We use the method of Goldsmith et al. (2010) for estimation of the parameters of the generalized functional linear model (1.3), called "Penalized Functional Regression" or PFR. This method represents the functional predictor using a Karhunen-Loeve expansion (principal component basis). It uses a large number of these bases, capturing nearly all the information in the functional data. In the resulting regression model, smoothing of the coefficient function $\beta(t)$ is used to enforce parsimony. This method is related to the functional regression framework of Cardot et al. (2003), Cardot and Sarda (2005), but improves upon it in a number of aspects, including: (1) the ability to handle functions that are observed with noise; (2) the connection to mixed effects models, which provides a framework for generalization, and a stable and efficient method for computation; (3) the automatic selection of the smoothing parameters. PFR consists of three steps: (1) smoothing the covariance matrix of the predictors; (2) obtaining principal components from the smoothed covariance matrix; (3) estimating $\beta(t)$ by penalized splines.

Consider the context where we have noisy observations $x_{i,t} = X_i(t) + \varepsilon_{i,t}$ of the functional predictor $X_i(t)$, where $\varepsilon_{i,t}$ is normally distributed with unknown variance σ_ε^2. Let $K^{X'}$ be the empirical covariance matrix of the vectors $\{(x_{i,1},\ldots,x_{i,T})^T\}_{i=1}^T$ and let $K^X(s,t) = Cov\{X_i(s), X_i(t)\}$ be the unknown covariance function of $X_i(t)$ defined on the domain $[0,T]^2$. PFR views the elements of $K^{X'}$ as noisy observations of the function $K^X(s,t)$ on the domain $[0,T]^2$, and estimates $K^X(s,t)$ in the following way: (1) drop the diagonal elements of the empirical covariance matrix $K^{X'}$, (2) obtain an estimate of $K^X(s,t)$ ($\hat{K}^X$) as a smoothed matrix of $K^{X'}$ with its diagonal elements removed.

PFR obtains the eigenvectors $\{\psi_k(t)\}$ of $\hat{K}^X$. These are the estimated principal com-

ponent functions for $\{X_i(t)\}_{i=1}^{N}$. Then we can obtain estimates of $X_i(t)$ as

$$\hat{X}_i(t) = \sum_{k=1}^{K_x} c_{i,k}\, \psi_k(t), \tag{4.1}$$

where $c_{i,k} = \sum_{k=1}^{T} x_{i,t}\, \psi_k(t)$ is the corresponding loading for $\psi_k(t)$. Here K_x is the truncation level; a distinguishing feature of PFR is that it takes K_x to be large, e.g. $K_x = 35$, rather than choosing K_x to capture a fixed percent of the variability in the functional data.

In order to estimate $\beta(t)$, PFR first represents $\beta(t)$ by expanding in a different basis. In particular, it uses a truncated power basis of degree 2, denoted by $\{\phi_k(t)\}_{k=1}^{K_b}$. Then

$$\beta(t) = \sum_{k=1}^{K_b} b_k\, \phi_k(t) = \boldsymbol{\phi}(t)\boldsymbol{b}, \tag{4.2}$$

for unknown $\boldsymbol{b} = \{b_1, \ldots, b_{K_b}\}^T$, and $\beta(t)$ is shrinking towards quadratic function in PFR. From (4.1)-(4.2), for the i^{th} server, (1.3) becomes

$$g(EW_i) = \alpha + \int_0^T \boldsymbol{c_i'}\, \boldsymbol{\psi}^T(t)\, \boldsymbol{\phi}(t)\, \boldsymbol{b}\, dt = \alpha + \boldsymbol{c_i'} \boldsymbol{J}_{\psi\phi} \boldsymbol{b}, \tag{4.3}$$

where $\boldsymbol{c_i'} = \{c_{i,1}, \ldots, c_{i,K_x}\}^T$ and $\boldsymbol{J}_{\psi\phi}$ is a $K_x \times K_b$ matrix with the (i,j) th entry $\int_0^T \psi_i(t)\, \phi_j(t)\, dt$.

To estimate $\boldsymbol{b}$, we first assume that $\boldsymbol{b} \sim N(0, \boldsymbol{D}^{-1})$, where $\boldsymbol{D}$ is a penalty matrix. Let $\boldsymbol{C}$ be the $N \times K_x$ matrix with the i^{th} row equal to $\boldsymbol{c_i'}$. Then, (1.3) becomes

$$\begin{aligned} g(E\boldsymbol{W}) &= [1\ \boldsymbol{C}\boldsymbol{J}_{\psi\phi}][\alpha\ \boldsymbol{b}]^T \\ \boldsymbol{b} &\sim N(0, \boldsymbol{D}^{-1}), \end{aligned} \tag{4.4}$$

where $\boldsymbol{W} = (W_1, \ldots, W_N)^T$ and $\boldsymbol{D}$ is a identity matrix with first three diagonal elements be zeros. This is the choice of $\boldsymbol{D}$ used in Goldsmith et al. (2010). Combining with the assumption that $W_i \sim EF(EW_i, \tau)$, i.e. that W_i is distributed according to an exponential family distribution with dispersion parameter τ, this yields a generalized linear mixed

effect model with parameters α, $\boldsymbol{b}$, τ (Ruppert et al. 2003), and $\boldsymbol{b}$ can be estimated using standard mixed effects software. We tried several estimation methods for $\boldsymbol{b}$, (1) gam function in R with "REML", (2) gam function in R with "GCV.cp", (3) MCMCglmm function in R. The best prediction result is obtained when we use gam funciont with "GCV.cp", so we only report the result from "GCV.cp" in this work. The estimated beta function is obtained by

$$\hat{\beta}(t) = \boldsymbol{\phi}(t)\hat{\boldsymbol{b}}, \tag{4.5}$$

To understand the impact of smoothing the predictors, we applied PFR with and without smoothing the covariance matrix of the predictors (step (1)), to the datasets from DryadLINQ, and observed almost identical results. Also, a quadratic extention for PFR (See Section 4.1.4) and Bayesian extention for PFR (See Section 4.1.5) are applied to our data, but seem to provide the similar results with the PFR. Hence, we only list results obtained from PFR with the smoothing step in this chapter.

4.1.2 A Simpler FDA Method

We also consider an alternative approach, which consists of (1) smoothing the predictor functions $X_n(t)$ individually; (2) Estimating $\beta(t)$ without smoothing. We do (1) using the truncated polynomial basis (Ruppert et al. 2003) to represent the predictor function $X(t)$, and the coefficients of this representation are estimated by penalized maximum likelihood. For each metric (e.g. CPU utilization) we obtain a common smoothing parameter to use when estimating the individual predictor functions. This is due to the fact that these functions can be viewed as coming from a single population, with common characteristics including the level of smoothness. To obtain a single value for the smoothing parameter, we first use Restricted Maximum Likelihood (REML) (Neu-

maier and Groeneveld 1998) to estimate the smoothing parameter for each functional observation; then we take the median of these values. After the principal component decomposition, we apply generalized linear regression to EW_n and principal component basis which capture 95% variability of data. $\beta(t)$ is then estimated by the product of regression coefficients and principal component basis.

4.1.3 Generalized Linear Model

We compare our FDA approaches with generalized linear regression where the discrete values $\{x_{n,t}\}_{t=1}^{T}$ are used as predictors (ignoring their time series structure). In the case of a binary outcome we use logistic regression, while for a continuous outcome we use linear regression.

4.1.4 Functional Quadratic Regression

An extention to the generalized function linear model (1.3) is the generalized functional quadratic regression (Yao and Müller 2010), where the dependency of a scalar response on a functional predictor is of quadratic rather than linear nature. The definition is

$$g(EW) = \alpha + \int_{\mathscr{C}} X(t)\,\beta(t)\,dt + \int_{\mathscr{C}}\int_{\mathscr{C}} X(s)\,X(t)\,\gamma(s,t)\,ds\,dt, \tag{4.6}$$

where α and $\beta(t)$ are the intercept and link function as usual, and $\gamma(s,t)$ is a square integrable bivariate quadratic parameter function associated with the quadratic term.

The applicaton of quadratic terms in PFR results in a generalization of (4.4), that is

$$\begin{aligned} g(E\boldsymbol{W}) &= [1\ \boldsymbol{CJ}\ \boldsymbol{CJ2}][\alpha\ \boldsymbol{b}\ \boldsymbol{r}]^T, \\ \boldsymbol{b} &\sim N(0, \boldsymbol{D}^{-1}), \\ \boldsymbol{r} &\sim N(0, \boldsymbol{D2}^{-1}), \end{aligned} \tag{4.7}$$

where $\boldsymbol{CJ}$ and $\boldsymbol{D}$ are the same in Section 4.1.1. $\boldsymbol{CJ2}$ is the $N \times 2K_b$ matrix with the $(i * K_b + j)^{th}$ column equal to $\langle \boldsymbol{CJ}_i, \boldsymbol{CJ}_j \rangle$, and $\boldsymbol{D2}$ is the $K_b \times K_b$ identity matrix.

4.1.5 Bayesian Penalized Functional Regression

Bayesian penalized functional regression is an alternative method to PFR by estimating α, $\boldsymbol{b}$, τ in PFR from a Bayesian perspective. Firstly, we obtain the same $\boldsymbol{CJ}_{\psi\phi}$ from the truncated polynomial basis in Section 4.1.1, and orthogonalize the covariance matrix of $\boldsymbol{CJ}_{\psi\phi}$ to get A, such that $A^T A = Cov(\boldsymbol{CJ}_{\psi\phi})$. Then (4.4) becomes

$$g(E\boldsymbol{W}) = [1\ \boldsymbol{CJ}_{\psi\phi} A][\alpha\ A^T \boldsymbol{b}]^T, \tag{4.8}$$

Since $Cov(\boldsymbol{CJ}_{\psi\phi} A)$ is a diagonal matrix, we have uncorrelated observations $\boldsymbol{CJ}_{\psi\phi} A$ here. Instead of estimating $\boldsymbol{b}$, we estimate $A^T \boldsymbol{b}$, denoted by $\boldsymbol{b'}$. Secondly, we assign priors to α, $\boldsymbol{b'}$, τ as follows.

$$\begin{aligned} \alpha &\sim N(0, 10^{10}), \\ \boldsymbol{b'} &\sim N(\boldsymbol{0}, P), \\ \tau &\sim Unif(0.01, m_\tau^2), \end{aligned}$$

where m_τ is the maximum entry of $\boldsymbol{b}$ estimated from (4.4).

4.2 Application to DryadLINQ

The methods described in Secion 4.1 are applied to datasets from DryadLINQ system of Microsoft (Isard et al. 2007, Yu et al. 2008). DryadLINQ is a high-performance, general-purpose distributed computing system that handles cluster-based distributed computing. It combines two important pieces of Microsoft technology: the Dryad distributed execution engine and the .NET Language Integrated Query (LINQ) (Institute 2011). DryadLINQ solves the major challenge of implementing a distributed application for large datasets, and is used routinely by Microsoft to analyze petabytes of data on clusters of thousands of computers. Here, we will analyze data from several different jobs assigned to the DryadLINQ system. The jobs we analyze are actually stages of a larger computation. Each stage has to be completed before the next stage can be started. So when one or more servers takes a very long time to finish one stage, it holds up the entire system from starting the next stage. In the following three datasets, the metrics for each server are measured every 10 seconds.

1. "Click Bot." This is a task to detect malignant or automatic clicks on on-line advertisements. The task was run using 441 servers, for which four metrics were collected: CPU, disk, memory, and network utilization. Among these servers, the majority had task run times less than 18. The DryadLINQ operators consider a server that will have a run time of greater than 18 to require early corrective action, so we predict whether or not the run time will exceed this tolerance threshold. Figure 4.1 shows the time series of CPU utilization for servers having run time ≤ 18, i.e. having $W_n = 0$, and for servers having run time > 18, i.e. having $W_n = 1$. It only shows the first 11 time periods, because we use information early in the process to do the prediction. For visualization purposes we only show 30 servers in each plot. Clearly the CPU utilization behaves very differently in the

two cases, and it is possible to predict W_n with a high degree of accuracy using the CPU utilization. We also examined the time series of memory utilization, as shown in Figure 4.2. We see that predicting W_n is very difficult in this scenario.

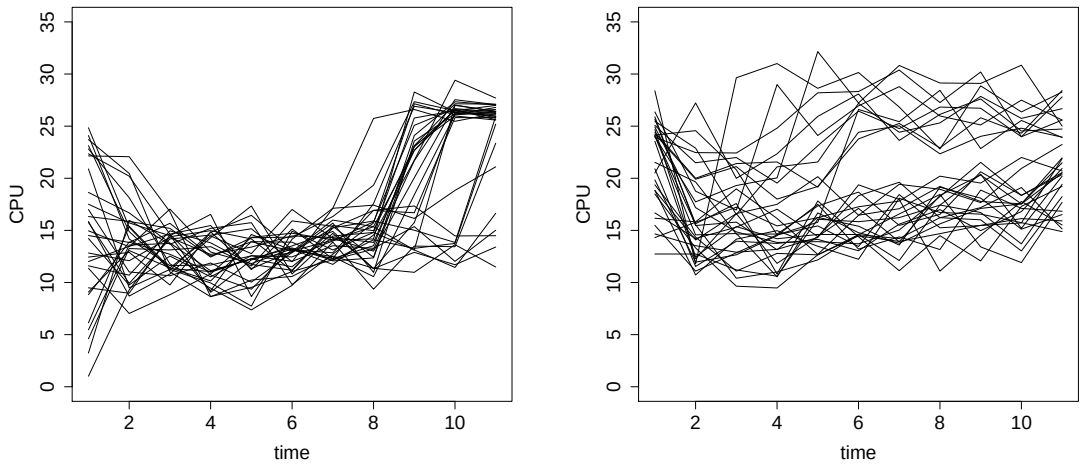

Figure 4.1: CPU utilization during the "Click Bot" job, for servers having run time below the tolerance level (left), and above the tolerance level (right).

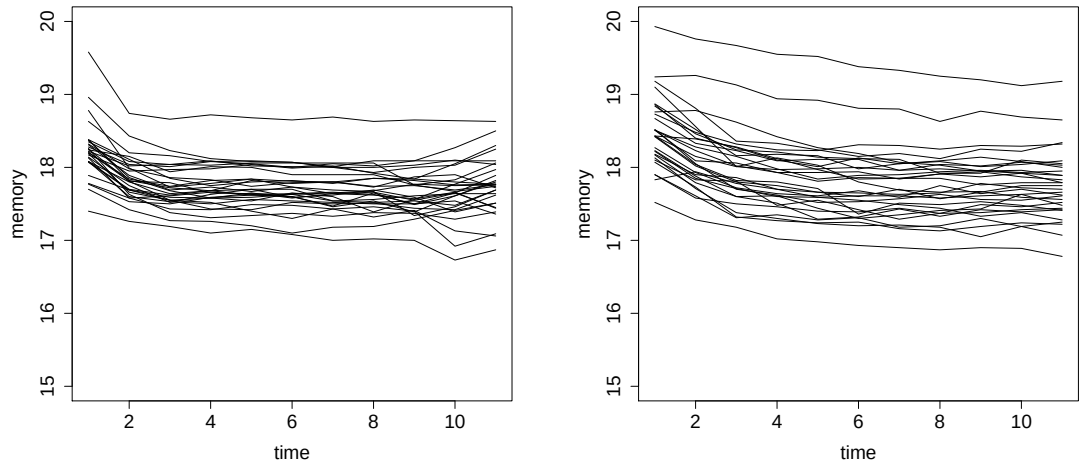

Figure 4.2: Memory utilization during the "Click Bot" job, for servers having run time below the tolerance level (left), and above the tolerance level (right).

2. "Tests Super." This refers to the tests for establishing the benchmarks of the DryadLINQ system. "Tests Super" is usually associated with assessing the performance characteristics of computer hardware and software. The task was run on 790 servers, for which CPU and memory utilization are collected, and the DryadLINQ operators set the tolerance level to be 10.

3. "Clue Web." This is a program for computing the basic statistics of a repository with approximately 1 billion web pages, e.g., the number of words per page, the number of links going out, etc. The task was run on 588 servers, for which CPU and memory utilization are collected. The tolerance level is set to be 6 by DryadLINQ operators.

4. "Bot Tracker." This (Waiting for Moises' answer). "Bot Tracker" was run on 239 servers with CPU and Disk utilization measured. The DryadLINQ operators are interested in identifying servers with run time larger than 100, so we choose the tolerance level 100.

4.3 Classification Application to DryadLINQ

In this section, we illustrate the accuracy of our methods in binary classification, and show the advantage of PFR against the other two methods. Recall that we are predicting whether the run time exceeds a specified tolerance d. We first apply them separately using each metric from the servers, and then apply them using all available metrics; the latter is done in practice.

We provide two approaches to evaluate the performance of classification: (1) Receiver Operating Characteristic (ROC) curve, which is a traditional way of measuring the accuracy of a binary classifier. If we claim $W_n = 1$ as positive and $W_n = 0$ as negative

for each node n ($n \in \{1,\ldots,N\}$), then ROC depicts the trade-off between False Positive (FP) and True Positive (TP); (2) Cost Curve with respect to the ROC curve, which measures the expected cost for each classification threshold value. Let $c_{0,1}$ be the cost of false positive, and $c_{1,0}$ be the cost of false negative. We first choose a sequence of values in the logarithmic scale for the thresholds. For each threshold value t, we obtain two misclassification costs $c_{0,1}$ and $c_{1,0}$, such that $c_{0,1} + c_{1,0} = 1$ and $c_{0,1}/c_{1,0} = e^t$. Then we calculate the realized cost $C = c_{0,1} * FP + c_{1,0} * FN$ for the threshold t. The cost curve serves as a prediction tool for estimating the log-odds surface as a function of the predictor. A vertical shift in the estimated log-odds curve does not affect the ROC curve, yet poor estimation of the vertical shift of the log-odds surface does affect the accuracy of cost-minimizing decision making, and this effect appears in the cost curve.

The classification and modeling procedure is described as follows. Recall that W_i is the indicator of whether the run time of the i^{th} node exceeds the threshold d, and we predict W_i using $x_{i,1},\ldots,x_{i,T}$ for $T < d$. To access the accuracy, we use the 3-fold cross-validation. We first divide the sets $\{W_i = 0\}$ and $\{W_i = 1\}$ into three even subsets, and combine each subset of $\{W_i = 0\}$ with a subset of $\{W_i = 1\}$. In this way, we have three subsets, each containing nodes from both $\{W_i = 0\}$ and $\{W_i = 1\}$. Choose one of the subsets to be the testing set, and the other two subsets to be the training set. Finally, we combine the log-odds from the three testing sets. This yields a predictive log-odds of $W_i = 1$ for each i in the dataset. Then we pick a sequence of values from -10 to 10 in the logarithmic scale for the thresholds, compare each threshold value to the predictive log-odds to calculate FP and TP, and obtain the single ROC curve and cost curve for this 3-fold cross-validation.

To summarize the information from the ROC and cost curves, we calculate the area under the curve (AUC). For the ROC curve, AUC is a traditional measure to compare

different classification methods, with value between 0 and 1. For the cost curve, AUC can be arbitrarily large depending on the accuracy of the methods. Basically, larger AUC is better for the ROC curve, while smaller AUC is better for the cost curve. We also report the cost values for five threshold points, namely $\{-5,-1,0,1,5\}$. Below we give results for prediction at a particular time period T for each dataset. We choose relatively small values for T to illustrate the utility of our methods for early prediction. The results are similar for other choices of T.

4.3.1 Results from "Click Bot"

Table 4.1 lists the summary results for three approaches we discussed in Section 4.1 when $T = 11$ and $d = 18$. In each approach, we have the binary classification on both the single-metric and the multi-metric cases. We find that PFR uniformly dominates the logistic regression and simple FDA in the ROC and the cost curves with larger AUC(ROC) and smaller AUC(Cost). For instance, PFR reduces roughly 30% of 1-AUC(ROC) and 25% of AUC(Cost) for the logistic regression in the single-metric cases. It also has smaller cost values for the five threshold points we choose. The poor performance of simple FDA in the memory metric is partially due to the flatness of their estimated $\hat{\beta}(t)$ functions, as defined in (4.5). Simple FDA chooses the smoothing parameter automatically, provides the flattest beta function among all methods, and can suffer over-smoothing. This occurs for the memory metric but not for the other metrics because the memory data is rather flat, and simple FDA smooths out nearly all the noises. Figure 4.4 shows three sets of estimated beta functions from the cross-validation for the processor and memory metrics. In the memory metric, the estimated beta functions from simple FDA are close to zero, which causes its under-performance in classification. Different from simple FDA, PFR doesn't smooth every sample series,

Table 4.1: Discriminative Methods: Binary Classification Results from "Click Bot" Dataset

Methods	Measures	CPU	Disk	Memory	Network	Multi
Logistic	AUC(ROC)	**0.9620**	**0.9698**	**0.8913**	**0.9653**	**0.9792**
	AUC(Cost)	**82.58**	**67.28**	**113.71**	**91.95**	**55.55**
	Cost(-5)	1.67	1.71	3.67	0.55	0.82
	Cost(-1)	15.42	9.42	26.42	13.11	12.61
	Cost(0)	13.50	12.00	23.50	18.50	13.50
	Cost(1)	12.03	11.50	15.14	12.84	8.19
	Cost(5)	1.25	0.23	0.41	2.29	0.28
FDA	AUC(ROC)	**0.9557**	**0.9730**	**0.6173**	**0.9771**	**0.9767**
	AUC(Cost)	**77.00**	**59.17**	**180.09**	**70.13**	**47.31**
	Cost(-5)	1.36	2.60	2.23	0.77	1.42
	Cost(-1)	16.15	10.96	50.59	10.19	8.54
	Cost(0)	16.50	10.00	36.50	13.50	9.00
	Cost(1)	10.99	8.03	18.75	10.92	6.92
	Cost(5)	1.28	0.20	0.45	1.37	0.30
PFR	AUC(ROC)	**0.9783**	**0.9782**	**0.9018**	**0.9759**	**0.9869**
	AUC(Cost)	**57.58**	**56.27**	**106.51**	**69.51**	**45.43**
	Cost(-5)	0.93	0.65	2.82	0.67	0.84
	Cost(-1)	10.61	10.96	23.03	11.65	9.34
	Cost(0)	12.00	12.00	22.00	12.50	12.00
	Cost(1)	9.57	7.57	14.30	11.72	6.65
	Cost(5)	0.27	0.23	0.42	1.38	0.29

which avoids the over-smoothing to some extent. Despite the potential weakness of over-smoothing, both FDA methods generally provide smoother and more interpretable curves than the logistic regression. For instance, in the case of the processor metric, both FDA methods provide similar estimated beta functions from the cross-validation. These $\hat{\beta}(t)$ functions can be interpreted as follows. The run time is positively associated with the CPU for the first few time periods and negatively correlated within the last few ones. The logistic regression, however, provide entirely different beta estimates in cross-validation, indicating its deficiency of overfitting. In the multi-metric case, both the ROC curve and the cost curve show the advantage of simple FDA and PFR, as shown in Figure 4.3. Specifically, PFR reduces 20% of 1-AUC(ROC) and 10% of AUC(Cost) compared with the logistic regression. The plots in other metrics can be found in the Appendix A.1.

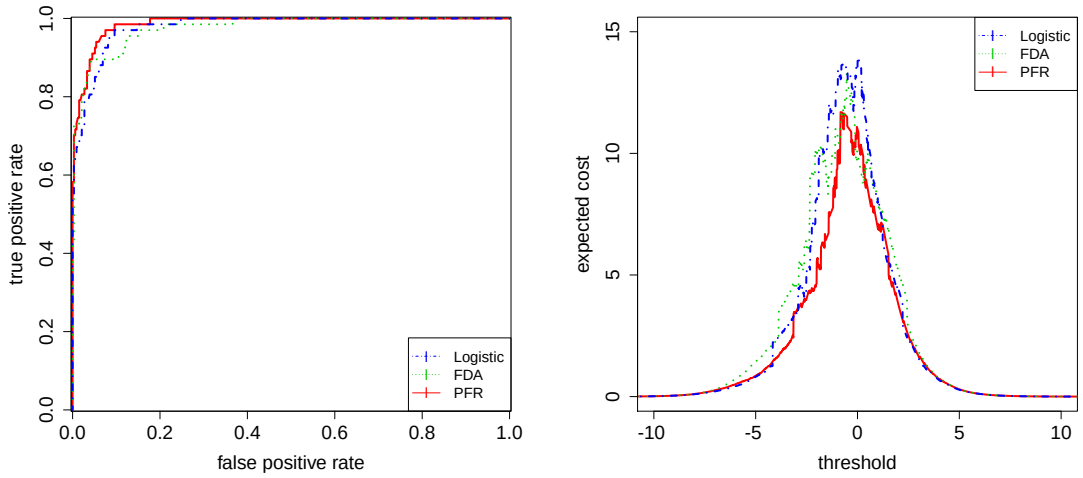

Figure 4.3: Discriminative methods: ROC Curve (left) and Cost Curve (right) from "Click Bot" Data in multi-metrics

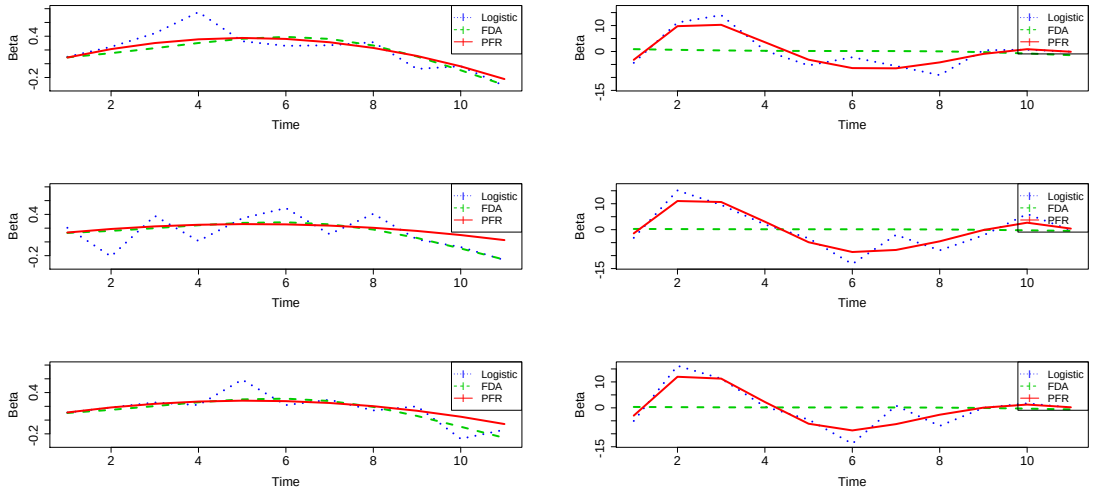

Figure 4.4: Estimated beta functions from "Click Bot" Data in the processor metric (left) and the memory metric (right) for the three cross-validation training sets

4.3.2 Results from other datasets

We also apply our methods to the other three datasets described in Section 4.2. For each dataset, only two single metrics are measured, so we list the summary results from them all together in Table 4.2. The time threshold T for "Tests Super", "Clue Web" and "Bot Tracker" reported here are 5, 4, 25 respectively.

1. In the "Tests Super" dataset, PFR reduces around 30% of 1-AUC(ROC) for the logistic regression for the individual metrics as well as the combined metric, and gets the best AUC(ROC) in all three cases. In terms of AUC(Cost), the logistic regression performs best in the CPU metric, but worst in the memory metric and multi metrics, while simple FDA obtains better result than the logistic regression and PFR by reducing roughly 20% AUC(Cost) for the logistic regression.

2. Simple FDA and PFR perform slightly better overall in the "Clue Web" dataset. However, all methods fail to accurately predict the difference of $W = 0$ and $W = 1$

Table 4.2: Binary Classification Results from Three Datasets

		Tests Super			Clue Web			Bot Tracker		
Methods	Measures	CPU	Memory	Multi	CPU	Memory	Multi	CPU	Disk	Multi
Logistic	AUC(ROC)	**0.9979**	**0.9836**	**0.9962**	**0.8619**	**0.5929**	**0.8671**	**0.5638**	**0.4157**	**0.4266**
	AUC(Cost)	**21.18**	**30.86**	**21.33**	**152.30**	**247.00**	**157.91**	**129.33**	**165.24**	**110.35**
	Cost(-5)	1.21	2.64	2.20	6.07	3.19	3.19	5.94	9.84	6.00
	Cost(-1)	3.54	4.00	3.00	31.42	66.02	33.24	23.22	24.34	23.19
	Cost(0)	3.00	2.50	2.50	28.00	45.00	28.50	19.50	18.50	16.00
	Cost(1)	1.54	1.81	2.54	15.60	24.47	15.60	12.96	12.30	9.65
	Cost(5)	1.03	0.05	0.03	0.39	0.61	0.43	1.15	1.16	0.17
FDA	AUC(ROC)	**0.9983**	**0.9899**	**0.9984**	**0.8651**	**0.5747**	**0.8690**	**0.5857**	**0.5002**	**0.5674**
	AUC(Cost)	**26.37**	**22.74**	**16.90**	**151.20**	**245.36**	**157.21**	**76.59**	**82.68**	**81.09**
	Cost(-5)	1.21	1.89	0.22	6.07	3.19	3.19	1.26	1.25	1.25
	Cost(-1)	4.08	3.73	2.54	31.42	61.71	37.78	18.62	21.92	20.31
	Cost(0)	3.50	3.00	2.50	28.50	45.50	28.50	12.50	12.5	13.5
	Cost(1)	3.73	1.08	2.27	15.60	24.47	15.60	6.72	6.72	6.72
	Cost(5)	1.02	0.05	0.02	0.39	0.61	0.43	0.17	0.17	0.17
PFR	AUC(ROC)	**0.9990**	**0.9872**	**0.9987**	**0.8681**	**0.5966**	**0.8799**	**0.5861**	**0.5011**	**0.5906**
	AUC(Cost)	**28.36**	**25.42**	**17.80**	**151.67**	**244.79**	**156.64**	**77.59**	**80.85**	**76.74**
	Cost(-5)	0.17	2.87	0.20	6.07	3.19	3.19	1.26	1.26	1.26
	Cost(-1)	2.81	3.73	2.00	31.96	64.48	35.90	18.89	19.23	17.35
	Cost(0)	3.50	3.00	2.50	28.00	45.50	29.00	13.00	12.50	12.50
	Cost(1)	3.46	1.81	2.27	15.33	24.47	15.60	6.72	6.72	6.72
	Cost(5)	2.00	0.05	0.02	0.41	0.61	0.42	0.17	0.17	0.17

for the memory metric. This is partially due to the lack of considerable difference between "regular" and "irregular" sets when we divide the data with tolerance $d = 6$ and make prediction based on just the first 4 time periods. Despite the inaccuracy, PFR performs a little better than the other two methods by reducing roughly 3% 1-AUC(ROC) and 1% AUC(Cost).

3. In the "Bot Tracker" dataset, predictive accuracy is poor for all three methods. The

poor performance is because of the high similarity between $W = 0$ and $W = 1$, as shown in Figure 4.5. However, both simple FDA and PFR exhibit a substantial advantage over the logistic regression by increasing roughly 35% AUC(ROC) while reducing around 40% AUC(cost) in all three cases. The big improvement for both FDA methods is partially due to the long predictor function with length 25. FDA methods generate smooth beta functions, which assign consistent weights to different portions of function predictor in cross-validation, while the logistic regression has three totally different beta estimates in cross-validation. An example of the processor metric is shown in Figure 4.5.

4.4 Prediction of the Run Time on DryadLINQ

In practice, operators in computing systems who want to know whether the servers will take extra time to finish their jobs may also want to predict their run time in advance. In this section, we apply linear regression and the two FDA methods described in Section 4.1 to the four datasets, and compare their prediction performance by means of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The prediction procedure is very similar to the one we described in Section 4.3, yet here we do not need to specify the tolerance level for separating the datasets into "regular" or "irregular". Instead, we only predict run time based on the metrics of the first T time periods.

4.4.1 Results from "Click Bot"

Figure 4.6 shows bar charts of RMSE and MAE in predicting the continuous run time of the "Click Bot" dataset. Each chart contains five bars, presenting results from each sin-

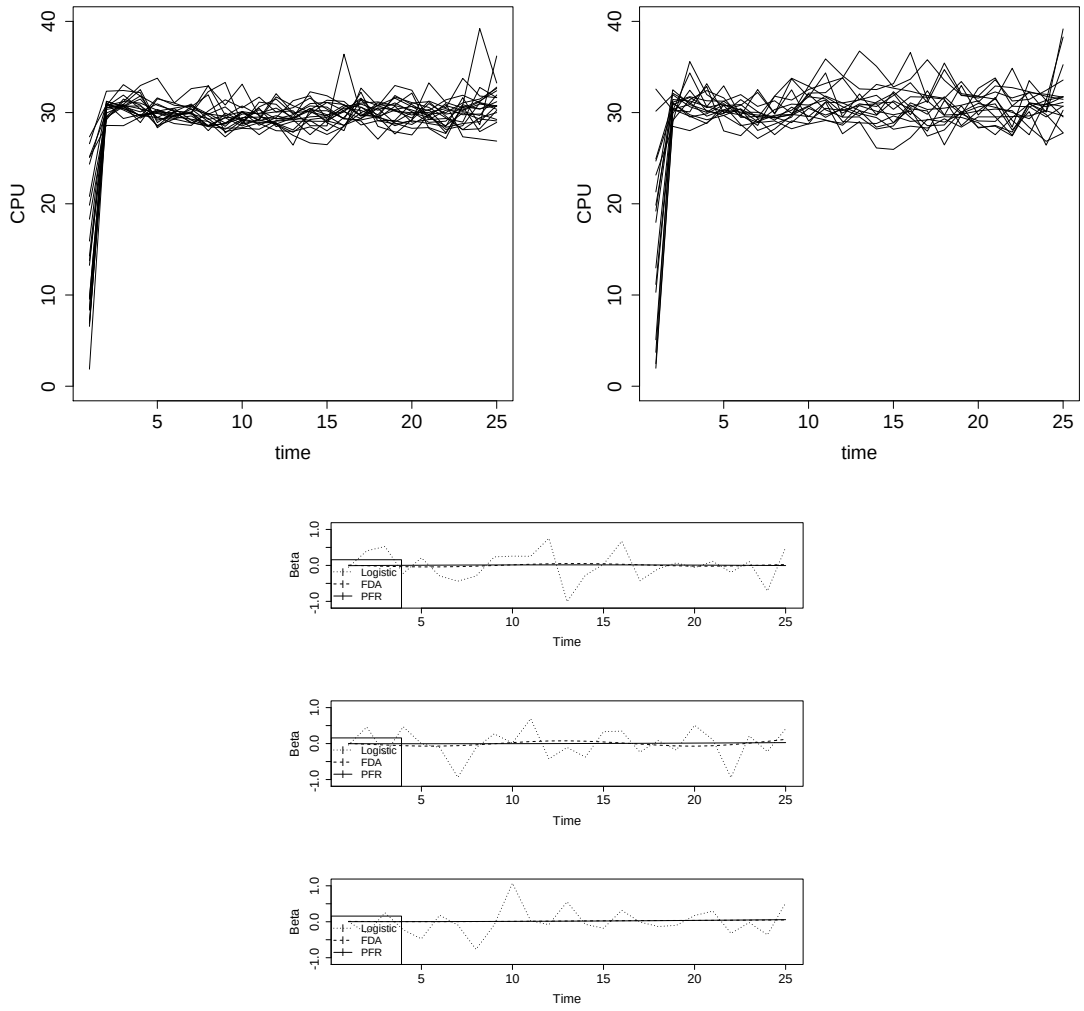

Figure 4.5: (1) CPU utilization during the "Bot Tracker" job, for servers having run time below the tolerance level (left), and above the tolerance level (right). (2) Examples of the estimated beta functions from "Bot Tracker" Data (bottom) for the three cross-validation training sets

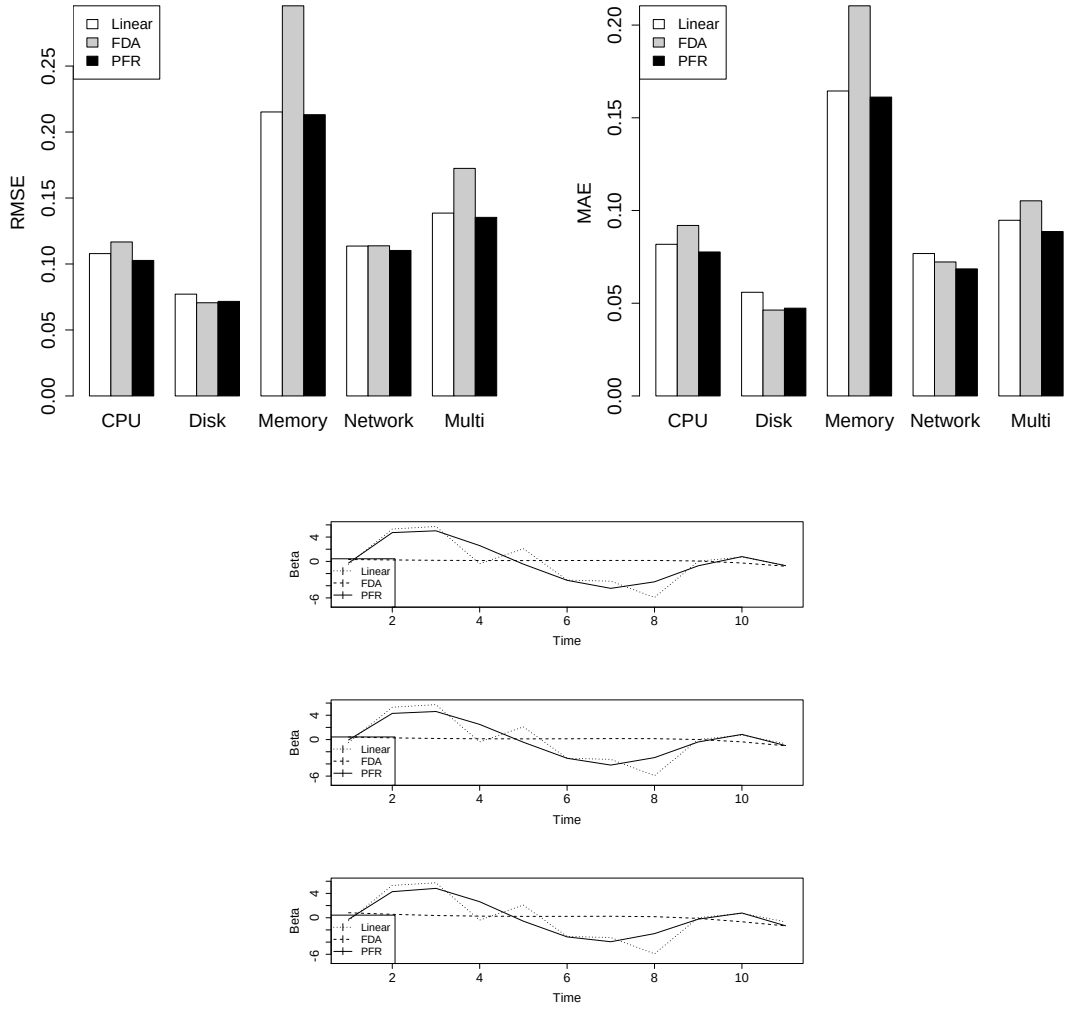

Figure 4.6: Root Mean Squared Error (left) and Mean Absolute Error (right) in "Click Bot" Data, and the estimate beta function in the memory metric in "Click Bot" Data (bottom)

gle metric, and all the four metrics, respectively. In each bar, we list the corresponding measures from the linear regression, simple FDA and PFR. As illustrated in Figure 4.6, simple FDA has a poor performance in predicting continuous run time, especially in the memory metric. This is due to the over-smoothing, a problem we also face in binary classification. Overall, PFR has a slight advantage over the linear regression by reducing roughly 5% of MAE and RMSE, as shown in Table 4.3. Also, PFR provides a more

Table 4.3: Continuous Prediction Results from "Click Bot" Dataset

Methods	Measures	CPU	Disk	Memory	Network	Multi
Logistic	MAE	**0.0818**	**0.0559**	**0.1645**	**0.0768**	**0.0947**
	RMSE	**0.1079**	**0.0771**	**0.2152**	**0.1136**	**0.1386**
FDA	MAE	**0.0919**	**0.0463**	**0.2104**	**0.0723**	**0.1052**
	RMSE	**0.1167**	**0.0706**	**0.2957**	**0.1138**	**0.1725**
PFR	MAE	**0.0776**	**0.0473**	**0.1612**	**0.0685**	**0.0887**
	RMSE	**0.1027**	**0.0717**	**0.2131**	**0.1102**	**0.1353**

interpretable $\hat{\beta}(t)$ function than the linear regression. In Figure 4.6, PFR tells us that (1) the run time is positively correlated with the memory metric in the first 5 time periods while negatively from 5 to 9 time periods; (2) The correlation magnitude increases for the first 3 periods, decreases from 3 to 7 time periods, and then increases again in the end. This interpretation is more reasonable than the result from the linear regression, which says the correlation is positive at first 3 periods, nearly zero at the 4^{th} period, positive again at the 5^{th} period and then becomes negative.

4.4.2 Results from other datasets

Table 4.4: Continuous Prediction Results from Three Datasets

		Tests Super			Clue Web			Bot Tracker		
Methods	Measures	CPU	Memory	Multi	CPU	Memory	Multi	CPU	Disk	Multi
Logistic	MAE	**0.0815**	**0.0943**	**0.0879**	**0.1722**	**0.1880**	**0.1801**	**0.4081**	**0.4418**	**0.4250**
	RMSE	**0.0989**	**0.1250**	**0.1126**	**0.2759**	**0.2930**	**0.2846**	**0.5323**	**0.5626**	**0.5477**
FDA	MAE	**0.1103**	**0.0911**	**0.1007**	**0.1701**	**0.1890**	**0.1796**	**0.3919**	**0.3930**	**0.3924**
	RMSE	**0.1403**	**0.1262**	**0.1334**	**0.2724**	**0.2938**	**0.2833**	**0.5131**	**0.5161**	**0.5146**
PFR	MAE	**0.0785**	**0.0922**	**0.0853**	**0.1680**	**0.1900**	**0.1790**	**0.3833**	**0.3948**	**0.3891**
	RMSE	**0.0981**	**0.1247**	**0.1122**	**0.2689**	**0.2946**	**0.2820**	**0.5108**	**0.5112**	**0.5110**

Table 4.4 summarizes the numerical results from the other three datasets, supporting the advantage of FDA methods against the linear regression, except for the memory metric in "Clue Web" dataset. Also, PFR shows the slightly improvement upon the simple FDA by reducing around 10%, 1%, and 3% of MAE and RMSE in "Tests Super", "Clue Web" and "Bot Tracker" respectively.

4.5 Conclusion

In this work, we compare two FDA approaches against traditional methods in binary classification (Section 4.3) and continuous prediction (Section 4.4). We show that both FDA methods perform uniformly better, and produce more interpretable results. However, simple FDA suffers over-smoothing of the beta functions when the data is flat, and works poorly in the continuous run time prediction. PFR universally outperforms the traditional methods, and provides consistent results in both cases. In conclusion, PFR is the approach we recommend for computing system management.

CHAPTER 5

GENERATIVE METHODS FOR PREDICTING PROCESS RUN TIME IN COMPUTING SYSTEMS

Large-scale distributed computing systems have become a fundamental part of computing infrastructure. Software applications are increasingly run in large parallel computing centers, from search and email to business applications like human resources management, accounting, and production scheduling. The growing scale of the computing centers leads to increasingly complex behavior, determined by the interaction of factors including workload, software, hardware, and external dependencies. These centers can no longer be managed via manual monitoring and control; for example, manual diagnosis of performance problems is no longer possible in real time, and requires enormous amounts of time and money when performed online. Thus, the real-time collection and analysis of data regarding system status and demand has become a necessary element in management of these centers. Specifically, the distributed computing system faces several challenges: (1) it is unclear how to do automatic debugging or profiling; and (2) although the computation is divided with the goal that the servers will finish at approximately the same time, this is not the case in practice. Due to network contention, differences in the storage location of data, and other problems that are not well understood, in fact some of the servers take much longer to finish, or do not finish at all. All these issues are difficult to diagnose in the absence of profiling tools. If one could recognize early that a particular server was going to take a long time to finish its assigned job, the job could be reassigned to an idle server, saving a substantial amount of time. Additionally, a statistical model relating server metrics (such as processor and disk utilization) to task completion time (which we discretize into very long or normal) could potentially help diagnose problems.

In Chapter 4, we discuss the performance of FDA methods for classifying servers in large-scale computing systems. We show that FDA provides more accurate and interpretable results. However, system operators are interested not only in predicting the process run time, but also in metric behaviors during specific time periods, e.g., how CPU utilization changes between normal and abnormal servers in the second minute of operations. FDA essentially treats the whole time series of metrics as a single entity, and takes their time dependence into account when building models. However, it does not capture the regime-switching behavior of the server data, where the transitions between the regimes occur at random times. Thus, it does not provide diagnostics information besides run time prediction. In this chapter, we fix this issue by proposing Hidden Markov/Semi-Markov Regime-Change Autoregression (HMRCA/HSMRCA), which models the regime-change behavior of servers. We show that HMRCA/HSMRCA are excellent tools for diagnosing computing systems.

HMRCA/HSMRCA are generative methods based on Hidden Markov/Semi-Markov models (HMM/HSMM) (Rabiner and Juang 1986, Elliott et al. 1995, Yu 2010). HMM models a Markov process with unobserved or hidden states, which are not directly visible. The outputs of the process, which depend on the states, however, are visible. The possible outputs of each state follows a probability distribution. Therefore, the sequence of observed outputs from HMM reveals partial information about the underlying states. HSMM models semi-Markov processes rather than Markov processes. Hence the probability of a regime-change depends on the sojourn time of the current state. This is in contrast to HMM, in which the probability of a regime-change is constant. To sum up, in HMRCA/HSMRCA: (1) The states follow a Markov/Semi-Markov process; (2) The states switch between different regimes; (3) Only the noisy outputs (system metrics) are visible; (4) Moreover, the regimes themselves follow an autoregressive process (Var 1998). In the following two sections, we formulate HMRCA and HSMRCA, and discuss

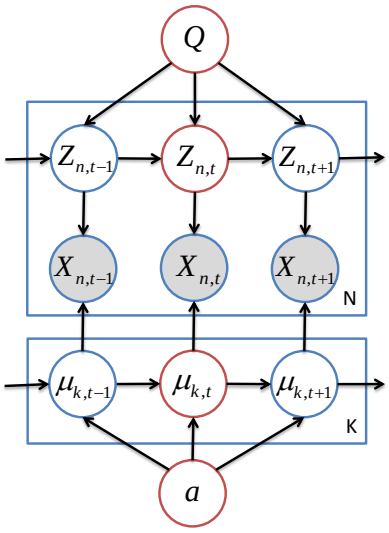

Figure 5.1: The graphical model of HMRCA

the estimation of both models.

5.1 Hidden Markov Regime-Change Autoregression

To understand the dependency of variables, we adopt a probabilistic graphical model to describe HMRCA (Figure 5.1). We use a graph to represent the conditional dependence structure between random variables. Many classical multivariate probabilistic models and methods, such as Bayesian statistics, pattern recognition, and machine learning can be studied under the diagram of general graphical models (Bishop et al. 2006). Probabilistic graphical models are graphs in which the nodes represent random variables, and the arcs represent conditional independence assumptions. As a result, they provide a compact representation of joint probability distributions (Murphy 2001).

In HMRCA, we have N sample time series of system metrics, like CPU utilization, shown as the upper rectangular in Figure 5.1. In this rectangular, $X_{n,t}$ is the noisy observation of the metric (hence it is shown shaded) for the n^{th} time series at time t, and $Z_{n,t}$

is the hidden regime indicator specifying which regime $X_{n,t}$ belongs to. The arrow from $Z_{n,t}$ to $X_{n,t}$ indicates that $X_{n,t}$ depends on $Z_{n,t}$. The arrows between $\{Z_{n,t}\}$ indicate that it is a Markov process, and the node Q above the rectangular governs the state transitions of $\{Z_{n,t}\}$. "N" in the right corner of this rectangular means that these dependencies are repeated N times. We also have K regimes, denoted by the lower rectangular in Figure 5.1. At time t, we assume that the k^{th} regime has a Gaussian distribution with mean $\mu_{k,t}$, and $\{\mu_{k,t}\}$ has a $AR(1)$ process with autoregressive coefficient a. We now formulate HMRCA as follows:

Hidden Markov Regime-Change Autoregression (HMRCA)

1. $X_{n,t} \mid Z_{n,t} = k \sim N(\mu_{k,t}, \sigma_k^2)$, $\forall k \in \{1,2,\ldots,K\}$, $\forall n \in \{1,2,\ldots,N\}$.
2. $Z_{n,t}$ follows a Markov chain with a $K \times K$ transition matrix $Q = \left(q_{i,j}\right)$, $\forall i \in \{1,2,\ldots,K\}$, $\forall j \in \{1,2,\ldots,K\}$.
3. $\mu_{k,t} = m(1-a) + a\mu_{k,t-1} + \varepsilon_k$, $\forall k \in \{1,2,\ldots,K\}$, where $\varepsilon_k \sim N(0, \sigma_\varepsilon^2)$, $a \sim Unif(-1,1)$, and m is the mean of $\{X_{n,t}\}$. Thus, the stationary distribution of $\mu_{k,t}$ is $N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$.
4. $\sigma_k \sim Unif(0,s)$, where s is the sample standard deviation of $\{X_{n,t}\}$.
5. $Z_{n,1}$ has a discrete uniform distribution over $\{1,\ldots,K\}$, i.e. $P\{Z_{n,1} = k\} = \frac{1}{K}$, $\forall n \in \{1,2,\ldots,N\}$.
6. $\mu_{k,1} \sim N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$, $\forall k \in \{1,2,\ldots,K\}$. The joint prior distribution for $\{\mu_{k,1}\}$ is $P(\{\mu_{k,1}\}) \propto \prod_{k=1}^{K} exp\left(-\frac{(\mu_{k,1}-m)^2(1-a^2)}{2\sigma_\varepsilon^2}\right) 1_{(\mu_{1,1}<\mu_{2,1}<\cdots<\mu_{K,1})}$.

We also make two assumptions to simply our model:

1. σ_k^2, the variance of the k^{th} regime, remains constant across all t.

2. Autoregressive coefficient a and autoregressive variance σ_ε^2 remain constant for all K regimes.

From Figure 5.1 and the above formulation, we know that the unknown parameters in HMRCA are

$$\theta = \left(Q, \{Z_{n,t}\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\right). \tag{5.1}$$

We use Gibbs sampling (Geman and Geman 1984) for estimating θ. Gibbs sampling is a Markov Chain Monte Carlo (MCMC) method (Neal 1993). MCMC constructs a Markov chain with the target distribution as its equilibrium distribution. After a large number of steps, samples from the Markov chain can be viewed as samples from the target distribution. MCMC methods are typically used to approximate target distributions in sophisticated probabilistic models, and are widely used in Bayesian statistics and machine learning. Gibbs sampling is one of the simplest MCMC methods. It samples a large set of random variables by sampling each variable separately, when their joint distribution is not known explicitly or difficult to directly sample from. In Gibbs sampling, we only need to sample from the conditional distribution of each variable. In HMRCA, we are able to write down the conditional distributions for all parameters. Hence, we choose Gibbs sampling for parameter estimation (Algorithm 6). For details, please refer to Appendix B.1.

In Algorithm 6, we sample every variable in $\theta = \left(Q, \{Z_{n,t}\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\right)$ subsequently in each iteration. They are sampled from their conditional distributions, denoted by $v \mid \cdot$, $v = Q$, $Z_{n,t}$, $\mu_{k,t}$, σ_k^2, a, or σ_ε^2. We set the iteration number $\mathscr{N} = 22000000$. To collect the samples, we choose the burn-in period to be 2000000, and the thinning period to be 1000, that is, we discard the first 2000000 samples from Gibbs sampling, and collect every 1000 samples afterwords. We choose these numbers to make their effective sample size (5.8) of all variables to be at least 1000. Since samples from MCMC

Algorithm 6: Gibbs Sampling for HMRCA

Choose initial values for $\theta = \left(Q, \{Z_{n,t}\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\right)$.

Initialization. Set iterator $i = 0$.

for $i \leq \mathscr{N}$ **do**

Sample each row of the transition matrix Q from a Dirichlet distribution.

$$Q_i \mid \cdot \sim Dir(\alpha_{i,1} + n_1, \ldots, \alpha_{i,K} + n_K).$$

Sample $Z_{n,t}$ from a discrete distribution with probability,

$$P(Z_{n,t} = k \mid \cdot) \propto \frac{q_{i,k}^{\delta_{\{t>1\}}} q_{k,j}^{\delta_{\{t<T\}}}}{\sigma_k} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right).$$

Sample $\mu_{k,t}$ from a normal distribution

$$\mu_{k,t} \mid \cdot \sim N(\mu, \sigma^2),$$

where μ and σ^2 are listed in (B.4) and (B.5).

Sample σ_k^2 from an inverse Gamma distribution

$$\sigma_k^2 \mid \cdot \sim inv - Gamma(\alpha_{\sigma_k^2}, \beta_{\sigma_k^2}),$$

where shape $\alpha_{\sigma_k^2}$ and scale $\beta_{\sigma_k^2}$ are listed in (B.7) and (B.8).

Sample a from a normal distribution

$$a \mid \cdot \sim N(\mu_a, \sigma_a^2),$$

where μ_a and σ_a^2 are listed in (B.10) and (B.11).

Sample σ_ε^2 from an inverse Gamma distribution

$$\sigma_\varepsilon^2 \mid \cdot \sim inv - Gamma(\alpha_{\sigma_\varepsilon^2}, \beta_{\sigma_\varepsilon^2}),$$

where shape $\alpha_{\sigma_\varepsilon^2}$ and scale $\beta_{\sigma_\varepsilon^2}$ are listed in (B.13) and (B.14).

end

are correlated, their effective sample size is a criterion for determining the quality of the samples. In our Gibbs sampling, the samples we collect can be viewed as 1000 independent samples from the target distribution – the joint distribution of θ in (5.1). In our implementation, we set number of regimes $K = 3$, and assume that the system metrics keep switching between them. We tried values from 2 to 5, and adopt $K = 3$ because it provided the best results. Regarding initial values, we choose Q to be a matrix with diagonal entries 0.8 and off-diagonal entries 0.1. We randomly generate $Z_{n,t}$ in $[1,3]$; $\mu_{k,t}$ between 25% and 75% quantiles of data; σ_k^2 between 0 and the sample variance of data, a between -1 and 1, and σ_ε^2 between 0 and 3.

Gibbs sampling is easy to implement, but suffers the convergence issue – it takes long time for the Markov chain to converge (Roberts and Smith 1994), especially when the probabilistic model is complex and contains many variables of time series (Albert and Chib 1993, Cowles and Carlin 1996). In (5.1), $\{Z_{n,t}\}$ follows a Markov chain with transition matrix Q, and $\{\mu_{k,t}\}$ follows an $AR(1)$ process. Since we sample these variables one at a time, the samples are highly correlated. To get a sufficient effective sample size, like 1000, we need a large number of iterations, burn-in period, and thinning period. Even we implement our models in fundamental programming languages, like C++, it takes approximately 30 minutes to finish Gibbs sampling. Based on the data description in Section 4.2, when we apply HMRCA to "Click Bot" with $K = 3$, we need to estimate 4941 variables. This makes us to think about a simpler model with fewer variables.

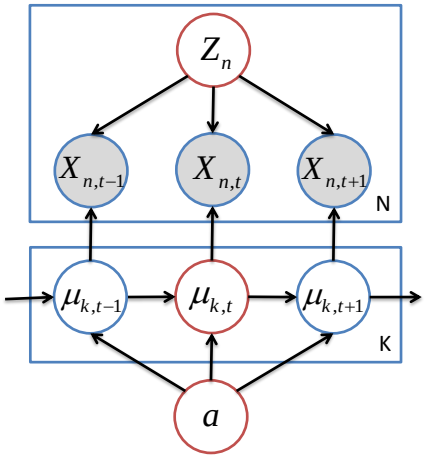

Figure 5.2: The graphical model of HMA

5.2 Hidden Markov Autoregression

In this section, we consider a parsimonious model that assigns only one regime indicator Z_n to each observation time series $\{X_{n,t}\}_{t=1}^{T}$. In this case, we have fewer variables to update in each iteration (just T regime indicators $\{Z_n\}$ instead of NT indicators $\{Z_{n,t}\}$) at the compensation of regime-change. Since this model does not capture regime change behavior, we name it Hidden Markov Autoregression (HMA). For the dependency of the model, please refer to Figure 5.2. All parameters and priors are the same as those described in Section 5.1, except that we do not have transition matrix Q for the regime-change. We now formulate HMA as follows:

Hidden Markov Autoregression (HMA)

1. $X_{n,t} \mid Z_n = k \sim N(\mu_{k,t}, \sigma_k^2)$, $\forall k \in \{1,2,\ldots,K\}$, $\forall n \in \{1,2,\ldots,N\}$.
2. $\mu_{k,t} = m(1-a) + a\mu_{k,t-1} + \varepsilon_k$, $\forall k \in \{1,2,\ldots,K\}$,
   where $\varepsilon_k \sim N(0, \sigma_\varepsilon^2)$, $a \sim Unif(-1,1)$, and m is the mean of $\{X_{n,t}\}$.
   Thus, the stationary distribution of $\mu_{k,t}$ is $N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$.

3. $\sigma_k \sim Unif(0,s)$, where s is the sample standard deviation of $\{X_{n,t}\}$.

4. Z_n has a discrete uniform distribution over $\{1,\ldots,K\}$, i.e.
   $P\{Z_n = k\} = \frac{1}{K}$, $\forall n \in \{1,2,\ldots,N\}$.

5. $\mu_{k,1} \sim N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$, $\forall k \in \{1,2,\ldots,K\}$. The joint prior distribution for $\{\mu_{k,1}\}$ is
   $P(\{\mu_{k,1}\}) \propto \prod_{k=1}^{K} exp\left(-\frac{(\mu_{k,1}-m)^2(1-a^2)}{2\sigma_\varepsilon^2}\right) 1_{(\mu_{1,1}<\mu_{2,1}<\cdots<\mu_{K,1})}$.

We are interested in the following parameters

$$\theta = \left(\{Z_n\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\right). \tag{5.2}$$

We describe the corresponding Gibbs sampling algorithm in Algorithm 7. For details, please refer to Appendix B.3. Compared with HMRCA, HMA dramatically reduces the number of variables. For instance, we only have 513 variables when applying HMA to "Click Bot" dataset, but we have 4941 variables in HMRCA. However, HMA fails to capture the regime-change pattern of data, and has a relative poor performance in classification (See Section 5.5). Thus, HMRCA is a better model in classifying abnormal servers, and we can not simply remove regime-change from HMRCA.

5.3 Hidden Semi-Markov Regime-Change Autoregression

We notice that a large portion of variables in HMRCA are $\{Z_{n,t}\}$, and they come from our assumption of the base model – HMM. A major drawback of HMM is the inflexibility in describing the time spent in a given regime. In HMM, the time of staying at a regime, also called sojourn time, follows a geometric distribution. However, we observe that servers usually stay at one regime for a period of time after entering it. Hence, we instead model the sojourn time using some other discrete distributions, like Poisson or negative binomial, because these distributions have higher probability for large values.

Algorithm 7: Gibbs Sampling for HMA

1 Choose initial values for $\theta = \left(\{Z_n\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\right)$.

2 **Initialization**. Set iteration number $\mathscr{N}$ and $i = 0$.

3 **for** $i \leq \mathscr{N}$ **do**

4 Sample Z_n from a discrete distribution with probability,

$$P(Z_n = k \mid \cdot) \propto \pi_k \prod_{t=1}^{T} \frac{1}{\sigma_k} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right). \quad \forall k \in \{1, 2, \ldots, K\}.$$

5 Sample $\mu_{k,t}$ from a normal distribution

$$\mu_{k,t} \mid \cdot \sim N(\mu, \sigma^2),$$

where μ and σ^2 are listed in (B.25) and (B.26).

6 Sample σ_k^2 from an inverse Gamma distribution

$$\sigma_k^2 \mid \cdot \sim inv - Gamma(\alpha_{\sigma_k^2}, \beta_{\sigma_k^2}),$$

where shape $\alpha_{\sigma_k^2}$ and scale $\beta_{\sigma_k^2}$ are listed in (B.28) and (B.29).

7 Sample a from a normal distribution defined in Line 8 of Algorithm 6.

8 Sample σ_ε^2 from an inverse Gamma distribution defined in Line 9 of Algorithm 6.

9 **end**

In this section, we describe HSMRCA, which is based on HSMM. Compared against HMM, HSMM provides a flexibility of modeling the sojourn time (Yu 2010). Figure 5.3 shows the graphical model of HSMRCA. In HSMRCA, we assume that there are N time series of observations $\{X_{n,t}\}$. Observations have their corresponding regime indicators $\{Z_{n,t}\}$. It differs from HMRCA (Figure 5.1) in that the n^{th} time series is divided into L_n segments ($n \in \{1, 2, \ldots, N\}$), where each segment has its own regime

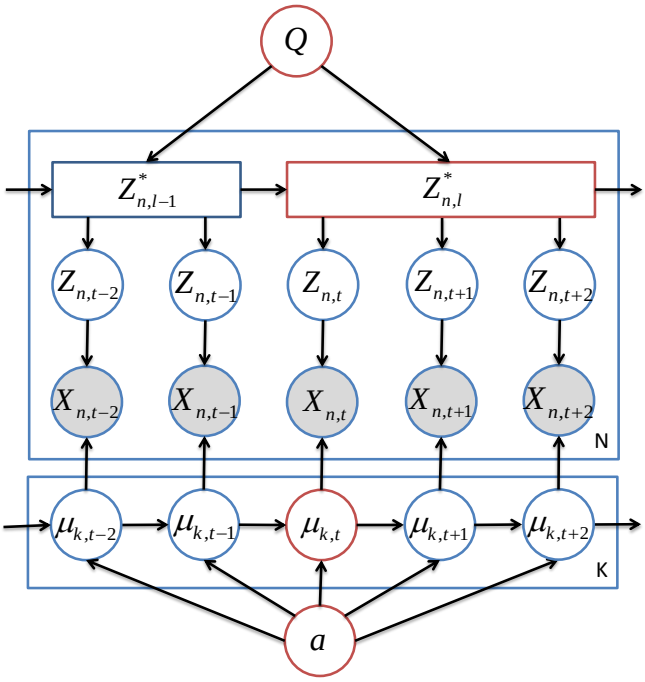

Figure 5.3: The graphical model for HSMRCA

indicator $\{Z^*_{n,l}\}$, denoted by the rectangles in Figure 5.3. We also assume that observations in each segment belong to the same regime. In Figure 5.3, arrows from $\{Z^*_{n,l}\}$ to $\{Z_{n,t}\}$ indicate that segment regime indicators decide individual regime indicators, and the arrows between $\{Z^*_{n,l}\}$ indicate that the regime-changes occur between segment regime indicators $\{Z^*_{n,l}\}$. Each segment has another two parameters $\tau_{n,l}$ and $t_{n,l}$, which are not shown in Figure 5.3. $\tau_{n,l}$ is the sojourn time of segment regime indicator $Z^*_{n,l}$, following a discrete distribution on $\{1,2,\ldots,\infty\}$. In HSMRCA, we assume a Poisson distribution for $\{\tau_{n,l}\}$. Since Poisson starts from 0 while sojourn time $\{\tau_{n,l}\}$ are positive, we define $\tau_{n,l}-1 \sim Poisson(\lambda)$. $t_{n,l}$ is the time of regime-switching for $Z^*_{n,l}$, i.e., $t_{n,l} = t_{n,l-1} + \tau_{n,l-1}$. Instead of updating $Z_{n,t}$ in HMRCA, we update $\{Z^*_{n,l}, \tau_{n,l}, t_{n,l}\}$ in HSMRCA.

Before we proceed to formulate HSMRCA, we first discuss its regime-change mechanism. In HSMRCA, we update segments in two steps. One is to adjust the length of

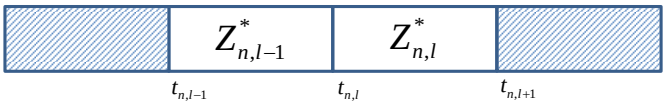

Figure 5.4: The small step of $\tau_{n,l}$

segments, which we call "small step"; the other is to change the number of segments by merging or splitting adjacent segments, which we call "large step". We discuss them separately in Section 5.3.1 and 5.3.2.

5.3.1 Small Step: Adjust Starting Time for Segments

To adjust the length of the l^{th} segment, it suffices to update its sojourn time $\tau_{n,l}$. It is equivalent to update its starting time $t_{n,l}$, since $t_{n,l}$ uniquely defines its adjacent sojourn time $\tau_{n,l}$ and $\tau_{n,l-1}$ ($\tau_{n,l} = t_{n,l+1} - t_{n,l}$ and $\tau_{n,l-1} = t_{n,l} - t_{n,l-1}$). Assume there are L segments, we only need to update $\{t_{n,l}\}$ for $l = 2, \ldots, L$, since $t_1 = 0$.

In Figure 5.4, we fix $t_{n,l-1}$ and $t_{n,l+1}$, and restrict $t_{n,l}$, such that $t_{n,l-1} < t_{n,l} < t_{n,l+1}$. Hence, there are $t_{n,l+1} - t_{n,l-1} - 1$ possible candidates for $t_{n,l}$, and we sample $t_{n,l}$ from a

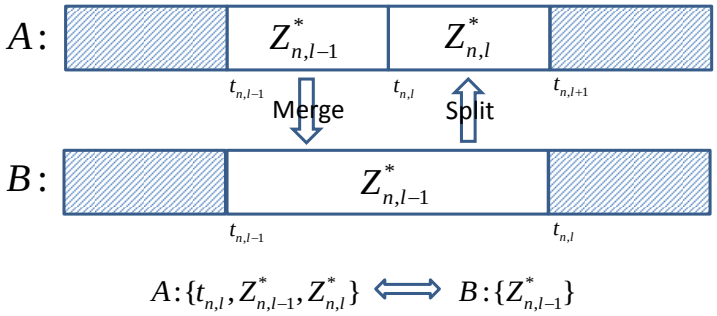

Figure 5.5: The large step of $\tau_{n,l}$

discrete distribution on $t' \in \{t_{n,l-1}+1, \ldots, t_{n,l+1}-1\}$ with probability

$$
\begin{aligned}
& P(t_{n,l} = t' \mid \{t_{n,s}\}_{s\neq l}, Z^*_{n,l-1} = i, Z^*_{n,l} = j, \{x_{n,t}\}, \{\mu_{k,t}\}, \{\sigma^2_k\}, Q) \\
& \propto P(\tau_{n,l-1} \mid t_{n,l})^{\delta_{\{t_{n,l}>1\}}} \times P(\tau_{n,l+1} \mid t_{n,l})^{\delta_{\{t_{n,l}<T\}}} \\
& \quad \times \prod_{t=t_{n,l-1}}^{t_{n,l}-1} P(X_{n,t} \mid Z_{n,t} = i, \mu_{i,t}, \sigma^2_i) \times \prod_{t=t_{n,l}}^{t_{n,l+1}-1} P(X_{n,t} \mid Z_{n,t} = j, \mu_{j,t}, \sigma^2_j) \\
& \propto \left(\frac{\lambda^{\tau_{n,l-1}-1}}{(\tau_{n,l-1}-1)!} \right)^{\delta_{\{t_{n,l}>1\}}} \times \left(\frac{\lambda^{\tau_{n,l}-1}}{(\tau_{n,l}-1)!} \right)^{\delta_{\{t_{n,l}<T\}}} \\
& \quad \times \prod_{t=t_{n,l-1}}^{t_{n,l}-1} exp\left(-\frac{(x_{n,t}-\mu_{i,t})^2}{2\sigma^2_i} \right) / \sigma_i \times \prod_{t=t_{n,l}}^{t_{n,l+1}-1} exp\left(-\frac{(x_{n,t}-\mu_{j,t})^2}{2\sigma^2_j} \right) / \sigma_j \\
& \propto \left(\frac{\lambda^{t'-t_{n,l-1}-1}}{(t'-t_{n,l-1}-1)!} \right)^{\delta_{\{t'>1\}}} \times \left(\frac{\lambda^{t_{n,l+1}-t'-1}}{(t_{n,l+1}-t'-1)!} \right)^{\delta_{\{t'<T\}}} \\
& \quad \times \prod_{t=t_{n,l-1}}^{t'-1} exp\left(-\frac{(x_{n,t}-\mu_{i,t})^2}{2\sigma^2_i} \right) / \sigma_i \times \prod_{t=t'}^{t_{n,l+1}-1} exp\left(-\frac{(x_{n,t}-\mu_{j,t})^2}{2\sigma^2_j} \right) / \sigma_j. \qquad (5.3)
\end{aligned}
$$

5.3.2 Large Step: Merge and Split Segments

Besides adjusting the position of $t_{n,l}$, we also introduce "merge and split" into the updating of $\tau_{n,l}$. As illustrated in Figure 5.5, we assume that all other segments remain the same, and there are two status for the current segment: (1) Status A, in which we have two segments and three undecided parameters $t_{n,l}$, $Z^*_{n,l-1}$, and $Z^*_{n,l}$ (note that $t_{n,l}$

and $\tau_{n,l}$ defines each other); (2) Status B, in which we have only one segment and one parameter, $Z^*_{n,l-1}$. The merge step goes from A to B, while the split step goes from B to A. However, it is not straightforward to write out the full conditionals for both status, so we can not use Gibbs sampling to estimate the parameters.

To solve this problem, we adopt the Metropolis-Hastings algorithm (Chib and Greenberg 1995). Metropolis-Hastings (MH) algorithm is the most commonly used MCMC method, for sampling from sophisticated, high-dimensional probability distributions (Neal 1993). In MH, we draw a proposed sample x' from any proposal density $P'(x \mid x^t)$ instead of the target density $P(x)$, where x^t is the parameter value at time t. We accept x' with an acceptance rate $\alpha = min\left\{\frac{P(x')P'(x^t|x')}{P(x^t)P'(x'|x^t)}, 1\right\}$, i.e., we set x^{t+1} equal to x' with probability α, and x^t otherwise. The advantage of MH is that we can sample from a simple proposal density P', even the target density P is complicated, or even not known.

In our updating scheme, we apply MH to a general "structural space", denoted by $\mathscr{S}$. In this space, each point is a configuration of the time series, for instance, both status A and B are considered as points in the space. The merge step (A to B) and the split step (B to A) are considered as moves in the space. Our target density P is the density of the whole space $\mathscr{S}$. We want to find a proposal density P', such that the acceptance rate for "split" is $\alpha_{split} = min\left\{\frac{P(A)P'(B|A)}{P(B)P'(A|B)}, 1\right\}$, and the acceptance rate for "merge" is $\alpha_{merge} = min\left\{\frac{P(B)P'(A|B)}{P(A)P'(B|A)}, 1\right\}$. In our design, we hope more merge or split attempts to be accepted, i.e., we need to find P' to make α as big as possible. A reasonable candidate for P' is the posterior distribution over the segment we are updating (the white blocks in Figure 5.5). Since $P(A)$ and $P(B)$ only differ in the configuration of the current segment, if we factor out $P(A)$ and $P(B)$, the probabilities of other segments cancel out, and hence $\frac{P(A)}{P(B)} = \frac{P'(A)}{P'(B)}$. In the split step, $\alpha_{split} = min\left\{\frac{P(A)P'(B|A)}{P(B)P'(A|B)}, 1\right\} = 1$, which means

that we always accept the split. Similarly, we always accept the merge.

In the split step (B to A), the new $t_{n,l}$ moves to a point between the current $t_{n,l-1}$ and $t_{n,l}$. After splitting, the first part becomes the new $Z^*_{n,l-1}$, the second part becomes the new $Z_{n,l}$, and the new $t_{n,l+1}$ moves to the current position of $t_{n,l}$. Before splitting, the parameters are $B : \{t_{n,l-1}, t_{n,l}, Z^*_{n,l-1} = k\}$. We need to decide the regimes for two new segments and new $t_{n,l}$ separating the two segments. There are $K \times K \times (t_{n,l} - t_{n,l-1} - 1)$ possible candidates for split parameters $A : \{t_{n,l-1}, t_{n,l}, t_{n,l+1}, Z^*_{n,l-1}, Z^*_{n,l}\}$. The proposal probability is then

$$
\begin{aligned}
&P'(A \mid B) = P'(Z^*_{n,l-1} = i, Z^*_{n,l} = j, t_{n,l} = t' \mid B) \\
&\propto P(Z^*_{n,l-1} = k \mid Z^*_{n,l-1} = i) \times P(Z^*_{n,l} = j \mid Z^*_{n,l-1} = k) \\
&\quad \times \prod_{t=t_{n,l-1}}^{t'-1} P(X_{n,t} \mid Z_{n,t} = i, \mu_{i,t}, \sigma_i^2) \times \prod_{t=t'}^{t_{n,l}-1} P(X_{n,t} \mid Z_{n,t} = j, \mu_{j,t}, \sigma_j^2) \\
&\propto q_{i,k} q_{k,j} \prod_{t=t_{n,l-1}}^{t'-1} exp\left(-\frac{(x_{n,t} - \mu_{i,t})^2}{2\sigma_i^2}\right) / \sigma_i \times \prod_{t=t'}^{t_{n,l}-1} exp\left(-\frac{(x_{n,t} - \mu_{j,t})^2}{2\sigma_j^2}\right) / \sigma_j. \quad (5.4)
\end{aligned}
$$

In the merge step (A to B), the new $t_{n,l}$ moves to the current position of $t_{n,l+1}$, and the merged segment is the new $Z^*_{n,l-1}$. Before we merge the segments, the parameters are $A : \{t_{n,l-1}, t_{n,l}, t_{n,l+1}, Z^*_{n,l-1} = i, Z^*_{n,l} = j\}$, and there are K candidates for $B : \{Z^*_{n,l-1}\}$. The proposal probability is then

$$
\begin{aligned}
&P'(B \mid A) = P'(Z^*_{n,l-1} = k \mid A) \\
&\propto P(Z^*_{n,l-1} = k \mid Z^*_{n,l-1} = i) \times P(Z^*_{n,l} = j \mid Z^*_{n,l-1} = k) \\
&\quad \times \prod_{t=t_{n,l-1}}^{t_{n,l+1}-1} P(X_{n,t} \mid Z_{n,t} = k, \mu_{k,t}, \sigma_k^2) \\
&\propto q_{i,k} q_{k,j} \prod_{t=t_{n,l-1}}^{t_{n,l+1}-1} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right) / \sigma_k. \quad (5.5)
\end{aligned}
$$

In the previous two sections, we have discussed the regime-change mechanism in

HSMRCA. We now sample different configurations of segments from the proposal probability defined in (5.5) and (5.4). A small step allows us to adjust the length of each segment, while a large step changes the number of segments. Yet how often should we take a large step? The frequency of large steps will not affect the validity of sampling, but will significantly impacts the convergence rate. Having tested different combinations in our implementation, we decide to take a large step every six iterations, that is, we take five small steps followed by a large step, either a mergence or a split. We now formulate HSMRCA as follows:

Hidden Semi-Markov Regime-Change Autoregression (HSMRCA)

1. $X_{n,t} \mid Z_{n,t} = k \sim N(\mu_{k,t}, \sigma_k^2)$, $\forall k \in \{1,2,\ldots,K\}$, $\forall n \in \{1,2,\ldots,N\}$.
2. $Z_{n,t} = Z^*_{n,l}$, if $t_{n,l} \leq t \leq t_{n,l+1}$, $\forall l \in \{1,2,\ldots,L\}$, $\forall n \in \{1,2,\ldots,N\}$.
3. $Z^*_{n,l}$ follows a Markov chain with a $K \times K$ transition matrix $Q = \left(q_{i,j}\right)$, $\forall i \in \{1,2,\ldots,K\}$, $\forall j \in \{1,2,\ldots,K\}$.
4. $\tau_{n,l} \sim Poisson(\lambda)$, $\forall l \in \{1,2,\ldots,L\}$, $\forall n \in \{1,2,\ldots,N\}$.
5. $\tau_{n,l}$ and $t_{n,l}$ is related by $\tau_{n,l} = t_{n,l+1} - t_{n,l}$ with $t_{n,1} = 0$, $\forall l \in \{1,2,\ldots,L\}$, $\forall n \in \{1,2,\ldots,N\}$.
6. $\mu_{k,t} = m(1-a) + a\mu_{k,t-1} + \varepsilon_k$, $\forall k \in \{1,2,\ldots,K\}$, where $\varepsilon_k \sim N(0, \sigma_\varepsilon^2)$, $a \sim Unif(-1,1)$, and m is the mean of $\{X_{n,t}\}$. Thus, the stationary distribution of $\mu_{k,t}$ is $N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$.
7. $\sigma_k \sim Unif(0,s)$, where s is the sample standard deviation of $\{X_{n,t}\}$.
8. $Z^*_{n,1}$ has a discrete uniform distribution over $\{1,\ldots,K\}$, i.e. $P\{Z^*_{n,1} = k\} = \frac{1}{K}$, $\forall n \in \{1,2,\ldots,N\}$.
9. $\mu_{k,1} \sim N(m, \frac{\sigma_\varepsilon^2}{1-a^2})$, $\forall k \in \{1,2,\ldots,K\}$. The joint prior distribution for $\{\mu_{k,1}\}$ is $P(\{\mu_{k,1}\}) \propto \prod_{k=1}^{K} exp\left(-\frac{(\mu_{k,1}-m)^2(1-a^2)}{2\sigma_\varepsilon^2}\right) 1_{(\mu_{1,1}<\mu_{2,1}<\cdots<\mu_{K,1})}$.

The unknown parameters in HSMRCA are

$$\theta = \left(Q, \{Z^*_{n,l}\}, \{\tau_{n,l}\}, \{\mu_{k,t}\}, \{\sigma^2_k\}, \lambda, a, \sigma^2_\varepsilon\right). \tag{5.6}$$

We adopt Metropolis-within-Gibbs (MWG) for estimating parameters (5.6) in HSMRCA. MWG is a hybrid MCMC method combining Metropolis-Hastings and Gibbs sampling (Tierney 1994). The initial idea is that it substitutes a Metropolis step when Gibbs sampling fails. In HSMRCA, we implement the large step (Section 5.3.2) using Metropolis-Hastings, and all the other estimations using Gibbs sampling. It is described in Algorithm 8. For details, please refer to Appendix B.5. HSMRCA combines the strengths of both HMRCA and HMA. It maintains the regime-switching property of HMRCA, while removing redundant regime-changes to simplify the model. For instance, we only have 2691 variables when applying HSMRCA to "Click Bot" dataset, which is around half of the number of variables in HMRCA.

5.4 Blocked Gibbs Sampling

In Section 5.2 and 5.3, we have proposed HMA and HSMRCA, which have fewer variables than HMRCA. In this section, we develop a novel estimation method, called "blocked Gibbs sampling", to improve the computational efficiency of HMRCA. Blocked Gibbs sampling was first proposed by Liu et al. (1994) as a variation of ordinary Gibbs sampling. Rather than sampling each variable individually, a blocked Gibbs sampling groups two or more variables together, and samples them from their joint conditional distribution. Recently, blocked Gibbs sampling has been widely used to improve the convergence rate of ordinary Gibbs sampling in many application areas (Ishwaran and James 2001, Tan and Hobert 2009, Yoon et al. 2010).

In HMRCA, Gibbs sampling exhibits slow mixing rates, because regime indicators

Algorithm 8: Metropolis-within-Gibbs for HSMRCA

Choose initial values for $\theta = \left(Q, \{Z^*_{n,l}\}, \{\tau_{n,l}\}, \{\mu_{k,t}\}, \{\sigma^2_k\}, \lambda, a, \sigma^2_\varepsilon\right)$.

Initialization. Set iteration number $\mathscr{N}$ and $i = 0$.

for $i \leq \mathscr{N}$ **do**

Sample each row of the transition matrix Q from a Dirichlet distribution.

$$Q_i \mid \cdot \sim Dir(\alpha_{i,1} + n_1, \ldots, \alpha_{i,K} + n_K). \quad \forall i \in \{1, 2, \ldots, K\}.$$

Sample $Z^*_{n,l}$ from a discrete distribution with probability,

$$P(Z_{n,t} = k \mid \cdot) \propto q_{i,k}^{\delta_{\{t_{n,l}>1\}}} q_{k,j}^{\delta_{\{t_{n,l}<T\}}} \prod_{t=t_{n,l}}^{t_{n,l+1}-1} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma^2_k}\right) / \sigma_k.$$

if $i\%5 \neq 0$ **then**

Merge the $l-1^{th}$ and l^{th} segments according to (5.5), or split the l^{th} segments according to (5.4);

else

Adjust $t_{n,l}$ for $\tau_{n,l}$ according to (5.3).

end

Sample λ from a Gamma distribution,

$$\lambda \mid \cdot \sim Gamma(\alpha + \sum_{n=1}^{N} \sum_{l=1}^{L_n} \tau_{n,l}, \sum_{n=1}^{N} L_n + \beta).$$

Sample $\mu_{k,t}$ from a normal distribution defined in Line 6 of Algorithm 6.

Sample σ^2_k from an inverse Gamma distribution defined in Line 7 of Algorithm 6.

Sample a from a normal distribution defined in Line 8 of Algorithm 6.

Sample σ^2_ε from an inverse Gamma distribution defined in Line 9 of Algorithm 6.

end

$\{Z_{n,t}\}$ are updated individually. This is fully discussed in Scott (2002) for finite HMM. Scott improved Gibbs sampling by incorporating the forward-backward algorithm (Rabiner 1989). The forward-backward algorithm is an efficient dynamic programming algorithm. It calculates conditional distributions of the hidden states given observed data and other parameters. Here we adopt this algorithm (described in Algorithm 9) for blocked Gibbs sampling, where we sample regime indicators $\{Z_{n,t}\}$ and regime means $\{\mu_{k,t}\}$ given observations $\{X_{n,t}\}$. In Line 5 of Algorithm 9, we sample $\{Z_{n,t}\}_{t=1}^{T}$ subsequently from the joint discrete distribution, calculated using forward-backward algorithm. In Line 6, we sample $\{\mu_{k,t}\}_{t=1}^{T}$ directly from their multivariate normal distribution. For details, please refer to Appendix B.2. Blocked Gibbs sampling for HMA is very similar to Algorithm 9, and is presented it in Appendix B.4.

Table 5.1: Effective sample sizes for selected variables in HMRCA

	Variables							
Methods	$q_{1,1}$	$q_{1,2}$	$\mu_{1,1}$	$\mu_{1,5}$	$\mu_{1,11}$	σ_1^2	a	σ_ε^2
Gibbs Sampling	18963	18311	406	297	317	344	392	493
Blocked Gibbs	19999	19999	2422	1668	1831	1987	2547	2554

We now compare the sampling efficiency of Gibbs sampling (Algorithm 6) and blocked Gibbs sampling (Algorithm 9) using the "Click Bot" dataset in DryadLINQ. In Section 5.1, we have noted that Gibbs sampling requires large iteration number and thinning period to achieve the necessary effective sample sizes for all variables in HMRCA. Here we change the iteration number from 22000000 to 2200000, burn-in period from 2000000 to 200000, and thinning period from 1000 to 100. We list partial simulation results in Table 5.4, where we give the effective sample sizes for selected variables in both methods. In Table 5.4, $q_{1,1}$ and $q_{1,2}$ are entries of the transition matrix Q; $\{\mu_{1,1}, \mu_{1,5}, \mu_{1,11}\}$ are means of the first regime at time 1, 5 and 11; σ_1^2 is the variance of

the first regime; a and σ_ε^2 are the coefficient and variance of the $AR(1)$ process. We observe that both methods obtain very large effective sample sizes for $q_{1,1}$ and $q_{1,2}$, which represent the transition probability between different regimes. Since there are only three regimes for "Click Bot", their values reach equilibrium very quickly. However, blocked Gibbs sampling samples more efficiently for all the other variables, with average effective sample size more than 1000, while the effective sample size from Gibbs sampling is less than 500. Because of its computational efficiency, we adopt blocked Gibbs sampling to estimate both HMRCA and HMA. Unless otherwise specified, all numerical results from HMRCA and HMA in this thesis are obtained from blocked Gibbs sampling.

5.5 Numerical Results from DryadLINQ

In this section, we compare all generative models in this chapter against penalized functional regression (Section 4.1.1). We apply these models to datasets "Click Bot", "Tests Super", "Club Web", and "Bot Tracker" from DryadLINQ (Section 4.2). We use blocked Gibbs sampling (Algorithm 9) for HMRCA and HMA, and Metropolis-within-Gibbs (Algorithm 8) for HSMRCA. For each MCMC algorithm, we run three chains with overdispersed starting points, and adopt Gelman and Rubin convergence diagnostics (Gelman and Rubin 1992) to evaluate their convergence. Specifically, we collect samples from three chains, calculate within-chain variance W and between-chain variance B. We then compute the potential scale reduction factor (PSRF),

$$\hat{R} = \sqrt{\frac{n-1}{n} + \frac{m+1}{mn}\frac{B}{W}}, \tag{5.7}$$

where n denotes the number of samples from each chain, and m denotes the number of chains. We keep sampling until $\hat{R}$ is less than 1.1. Finally, we combine mn samples from m chains, and calculate the effective sample size (Lenth 2001), i.e., the equivalent

Algorithm 9: Blocked Gibbs Sampling for HMRCA

1 Choose initial values for $\theta = \big(Q, \{Z_{n,t}\}, \{\mu_{k,t}\}, \{\sigma_k^2\}, a, \sigma_\varepsilon^2\big)$.

2 **Initialization**. Set iteration number $\mathscr{N}$ and $i = 0$.

3 **for** $i \leq \mathscr{N}$ **do**

4 Sample each row of the transition matrix Q from a Dirichlet distribution defined in Line 4 of Algorithm 6.

5 Sample $Z_{n,1}$ and $\{Z_{n,t}\}$, $\forall t \in \{2, \ldots, T\}$, from a discrete distribution with probability,

$$P(Z_{n,1} = k \mid \cdot) \propto \frac{\pi_k}{\sigma_k} exp\big(-\frac{(x_{n,1} - \mu_{k,1})^2}{2\sigma_k^2}\big)\, \beta_k(1),$$
$$P(Z_{n,t} = k \mid \cdot) \propto \frac{q_{i,k}}{\sigma_k} exp\big(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\big)\, \beta_k(t),$$

where $\beta_k(t)$ is defined in Appendix B.2.2.

6 Sample $\{\mu_{k,t}\}$ from a multivariate normal distribution,

$$\{\mu_{k,t}\}_{t=1}^{T} \mid \cdot \sim N(c, \Sigma),$$

where c and Σ are defined in Appendix B.2.1.

7 Sample σ_k^2 from an inverse Gamma distribution defined in Line 7 of Algorithm 6.

8 Sample a from a normal distribution defined in Line 8 of Algorithm 6.

9 Sample σ_ε^2 from an inverse Gamma distribution defined in Line 9 of Algorithm 6.

10 **end**

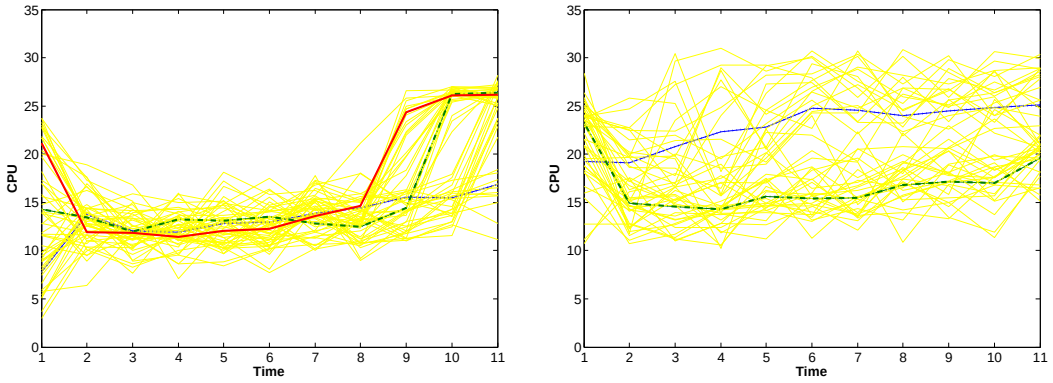

Figure 5.6: Means of Regimes from HMRCA for CPU utilization during the "Click Bot" job, for servers having run time below the tolerance level (left), and above the tolerance level (right).

number of independent samples,

$$M_{eff} = \frac{nm}{1+2\sum_{i=1}^{\infty}\rho_i}, \tag{5.8}$$

where ρ_i is the autocorrelation function at lag i. For each variable in our models, we require $M_{eff} > 1000$, and $\hat{R} < 1.1$.

We employ the same modeling and evaluation procedures as in Section 4.3:

- Adopt 3-fold cross-validation to obtain training and testing sets.
- Use ROC curve and cost curve to evaluate the performance of classification.
- Calculate the area under the curve (AUC) to summarize the information in ROC and cost curves.

Figure 5.6 shows the means of five regimes when applying HMRCA to time series of CPU utilization from "Click Bot", with three regimes for normal servers and two regimes for abnormal servers. The regimes capture the main structure of time series data as we expect. Each server belongs to only one regime, and keeps switching between

different regimes, as shown by CPU utilization metric. This is apparent for abnormal servers, because we observe many jumps and falls between two regimes. Figure 5.6 also demonstrates that generative methods like HMRCA and HSMRCA provide additional information for diagnostics of computing systems. For instance, system operators can analyze the regime-switching time for servers, and find the corresponding server status.

Table 5.2 summarizes the results of PFR and three generative methods on "Click Bot" dataset of DryadLINQ. We see that HMRCA and HSMRCA achieve similar accuracy as PFR, but have difficulty in classifying dataset from memory metric. For instance, both HMRCA and HSMRCA have 10% higher AUC(Cost) in the memory metric case. This is partially because normal and abnormal servers have similar regime-change patterns with different magnitudes (See Figure 4.2). HMA, being a parsimonious model without regime-change, performs poorly in classification. As illustrated in Table 5.2, both HMRCA and HSMRCA reduce roughly 20% of 1-AUC(ROC) and 30% of AUC(Cost) for HMA. This indicates that the regime-change design is effective in modeling the behavior of server metrics. Figure 5.7 plots the ROC curve and the cost curve of the above methods when applied to multi-metrics of "Click Bot". Both plots demonstrate the advantage of HMRCA and HSMRCA over HMA. For these curves in single-metric case, please refer to Appendix C.1.

Table 5.2: Generative Methods: Binary Classification Results from "Click Bot" Dataset

Methods	Measures	CPU	Disk	Memory	Network	Multi
PFR	AUC(ROC)	**0.9783**	**0.9782**	**0.9018**	**0.9759**	**0.9869**
	AUC(Cost)	**57.58**	**56.27**	**106.51**	**69.51**	**45.43**
	Cost(-5)	0.93	0.65	2.82	0.67	0.84
	Cost(-1)	10.61	10.96	23.03	11.65	9.34
	Cost(0)	12.00	12.00	22.00	12.50	12.00
	Cost(1)	9.57	7.57	14.30	11.72	6.65
	Cost(5)	0.27	0.23	0.42	1.38	0.29
HMA	AUC(ROC)	**0.9620**	**0.9350**	**0.8165**	**0.9670**	**0.9615**
	AUC(Cost)	**73.54**	**113.90**	**154.63**	**78.50**	**73.92**
	Cost(-5)	1.13	2.22	2.22	1.29	1.63
	Cost(-1)	13.80	23.42	39.78	15.61	14.19
	Cost(0)	14.50	24.00	33.50	17.00	16.50
	Cost(1)	10.72	14.52	18.01	12.14	10.14
	Cost(5)	0.29	0.44	0.44	1.39	0.32
HMRCA	AUC(ROC)	**0.9799**	**0.9755**	**0.8864**	**0.9768**	**0.9870**
	AUC(Cost)	**56.27**	**69.93**	**116.67**	**68.68**	**46.43**
	Cost(-5)	0.85	2.10	2.20	0.73	0.65
	Cost(-1)	11.61	13.84	27.38	10.10	10.73
	Cost(0)	12.50	14.00	23.50	12.50	12.50
	Cost(1)	9.37	10.48	14.52	11.45	6.45
	Cost(5)	0.28	0.41	0.44	1.38	0.30
HSMRCA	AUC(ROC)	**0.9789**	**0.9777**	**0.8983**	**0.9760**	**0.9869**
	AUC(Cost)	**56.91**	**58.83**	**116.85**	**70.55**	**45.79**
	Cost(-5)	0.85	1.40	2.23	0.68	0.69
	Cost(-1)	11.61	11.73	25.92	11.18	10.00
	Cost(0)	12.50	14.00	24.00	13.00	12.50
	Cost(1)	9.57	8.26	15.06	11.72	6.72
	Cost(5)	0.26	0.31	0.44	1.38	0.30

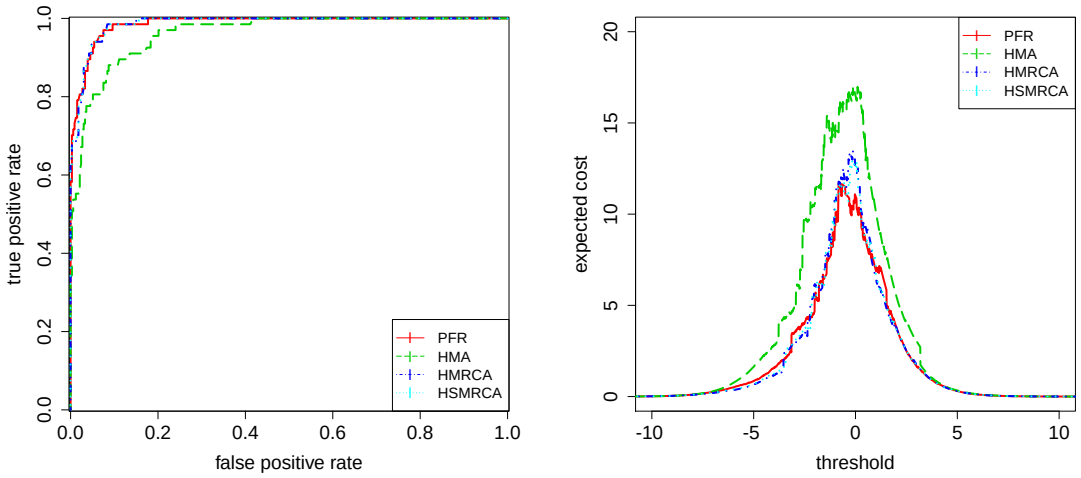

Figure 5.7: Generative methods: ROC Curve (left) and Cost Curve (right) from "Click Bot" Data in multi-metrics

CHAPTER 6

CONCLUSIONS

In this thesis, we conduct sparse principal component analysis on large matrix and tensor estimation, and provide diagnostics of distributed computing systems. Our main contributions are as follows.

We have developed a modified curvilinear algorithm for solving the eigenvalue problem on high dimensional matrices. We improve its computational efficiency in two aspects: (1) matrix inversion prevention using the Sherman-Morrison-Woodbury theorem, (2) convergence acceleration using a non-monotone search with Barzilai-Borwein step size instead of gradient descent, making it applicable for large matrices. We then incorporate this algorithm with an augmented Lagrangian method into a hybrid algorithm, and test it on randomly generated matrices against several benchmarking SPCA methods. We observe that the proposed hybrid algorithm has better numerical performance. When applied to a mean-reverted statistical arbitrage strategy, our algorithm significantly reduces transaction costs and provides more interpretable portfolios than other existing methods (e.g., standard PCA and exchange traded funds). Besides, the algorithm is also successfully implemented on three formulations of low-rank tensor recovery, by ??? alternating direction method of multipliers.

We consider diagnostics of distributed computing systems, and provide two distinct sets of solutions out of discriminative and generative methodologies, respectively. Under a discriminative framework, we compare the performance of penalized functional regression, functional additive model, and logistic regression, in predicting server runtime. We show that penalized functional regression is relatively easier to implement in real large-scale systems, and provides more accurate and interpretable results. In addition to identifying abnormal servers, the system operators are usually also interested

in discovering server behaviors during specific time periods. To achieve this goal, we formulate three generative models to capture the regime-switching behaviors of data. To improve the estimation efficiency of the models, we also design an innovative blocked Gibbs sampler based on the forward/backward algorithm. Our work performed well on real datasets from Microsoft DyradLINQ, and has been put into use in real computing systems.

Big data has attracted substantial attention over the last decade. People have long been talking about the opportunities and challenges it brings. The reason why we always focus on the computational capabilities of our models and algorithms on big datasets is that we face totally different problems as the data size scales up. Given the limited amount of existing literature in the area, we are pleased to offer a first step towards the scientific modeling of large-scale computing systems using modern statistics.

APPENDIX A

APPENDIX OF CHAPTER 4

A.1 Discriminative Methods: Binary Classification in "Click Bot"

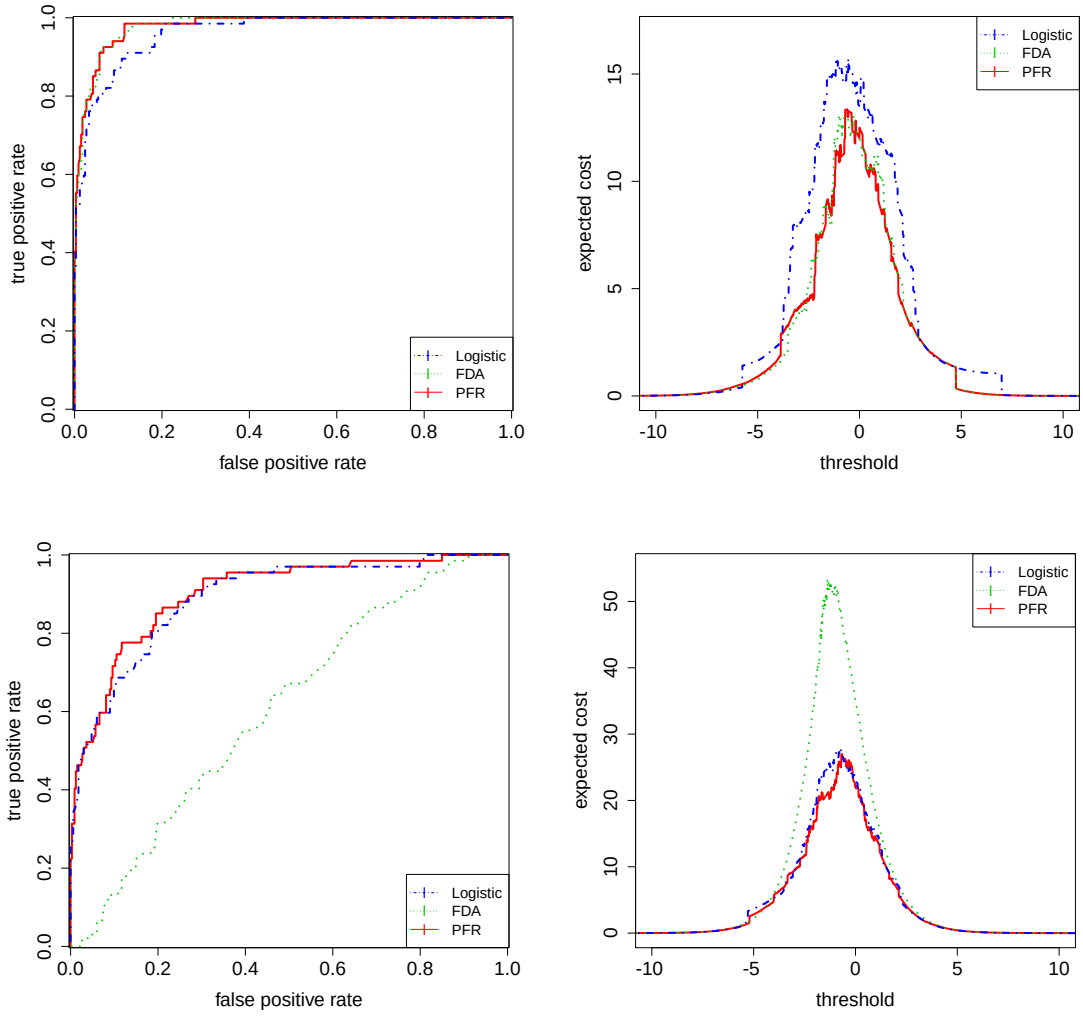

Figure A.1: Discriminative methods: ROC Curve (upper left) and Cost Curve (upper right) from "Click Bot" Data in processor metric, ROC Curve (lower left) and Cost Curve (lower right) from "Click Bot" Data in memory metric

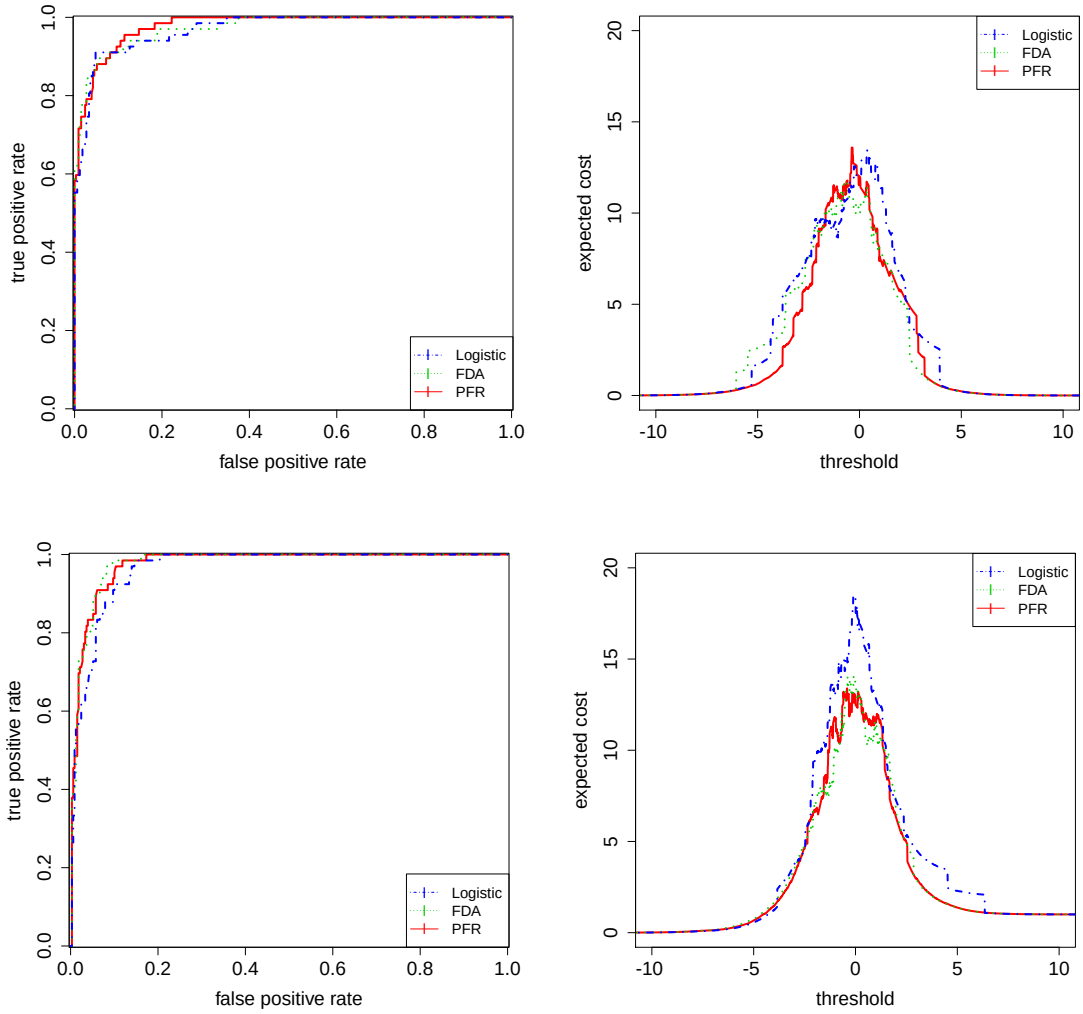

Figure A.2: Discriminative methods: ROC Curve (upper left) and Cost Curve (upper right) from "Click Bot" Data in disk metric, ROC Curve (lower left) and Cost Curve (lower right) from "Click Bot" Data in network metric

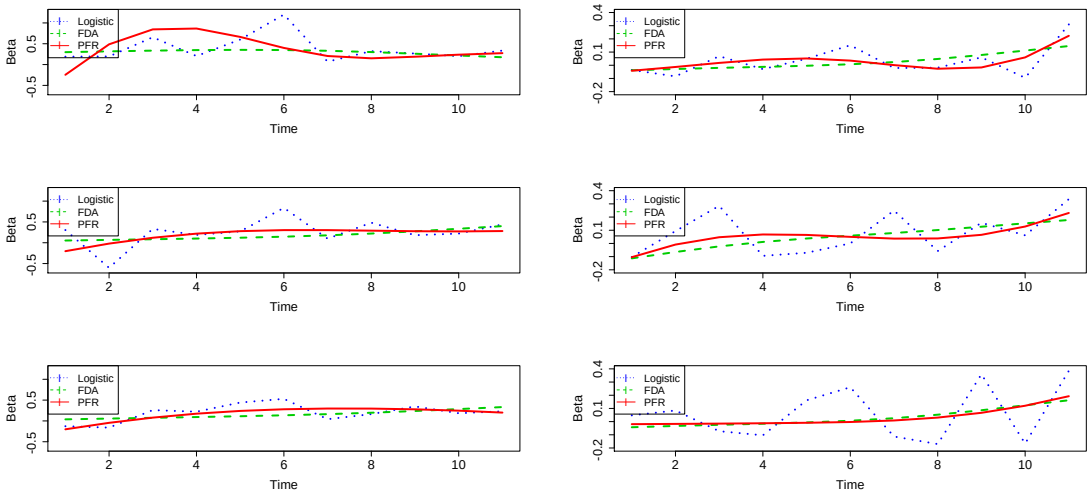

Figure A.3: Estimated beta functions from "Click Bot" Data in the disk metric (left) and the network metric (right) for the three cross-validation training sets

APPENDIX B

APPENDIX OF CHAPTER 5

B.1 Gibbs Sampling for HMRCA

B.1.1 Transition Matrix Q

Set $n_k = \#\{Z_{n,t} : Z_{n,t-1} = i, Z_{n,t} = k, \forall n \in \{1,2,\ldots,N\}\}$, and $Q_i = (q_{i,1},\ldots,q_{i,K})$, i.e. $Q = (Q_1,\ldots,Q_K)^T$. Suppose that the prior for Q_i is $Q_i \sim Dir(\alpha_i)$, where α_i is a $1 \times K$ row vector with the i^{th} entry to be 500, and all the other entries to be1. From Figure B.1,

$$\begin{aligned} P(Q \mid \{Z_{n,t}\}) &\propto \prod_{i=1}^{K} P(Q_i) \prod_{n=1}^{N} \prod_{t=2}^{T} P(Z_{n,t} \mid Z_{n,t-1}, Q) \\ &\propto \prod_{i=1}^{K} \prod_{k=1}^{K} q_{i,k}^{\alpha_{i,k}-1} \prod_{i=1}^{K} \prod_{k=1}^{K} P(Z_{n,t} = k \mid Z_{n,t-1} = i, Q_i)^{n_k} \\ &\propto \prod_{i=1}^{K} \left(\prod_{k=1}^{K} q_{i,k}^{\alpha_{i,k}-1} \prod_{k=1}^{K} q_{i,k}^{n_k} \right) \propto \prod_{i=1}^{K} \left(\prod_{k=1}^{K} q_{i,k}^{\alpha_{i,k}+n_k-1} \right). \end{aligned}$$

Hence the posterior distributions of $Q_i \mid \{Z_{n,t}\}$, $\forall i \in \{1,2,\ldots,K\}$ are conditionally independent, and $P(Q_i \mid \{Z_{n,t}\}) \propto \prod_{k=1}^{K} q_{i,k}^{\alpha_{i,k}+n_k-1}$, i.e.,

$$Q_i \mid \{Z_{n,t}\} \sim Dir(\alpha_{i,1}+n_1,\ldots,\alpha_{i,K}+n_K). \quad \forall i \in \{1,2,\ldots,K\}. \tag{B.1}$$

Note that we choose a Dirichlet prior for Q_i is because of the conjugacy.

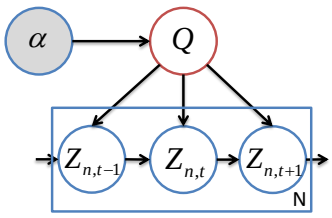

Figure B.1: The local graphical model for Q

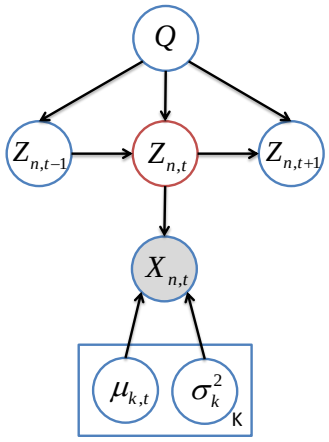

Figure B.2: The local graphical model for $Z_{n,t}$

B.1.2 Regime Indicator $Z_{n,t}$

We update $Z_{n,t}$ for $n \in \{1,2,\ldots,N\}$ and $t \in \{1,2,\ldots,T\}$, where we assume $Z_{n,t-1} = i$ and $Z_{n,t+1} = j$. From Figure B.2, we know

$$
\begin{aligned}
&P(Z_{n,t} = k \mid Z_{n,t-1} = i, Z_{n,t+1} = j, x_{n,t}, \mu_{k,t}, \sigma_k^2, Q) \\
&\propto P(Z_{n,t} = k \mid Z_{n,t-1} = i, Q)^{\delta_{\{t>1\}}} P(Z_{n,t+1} = j \mid Z_{n,t} = k, Q)^{\delta_{\{t<T\}}} P(X_{n,t} \mid Z_{n,t} = k, \mu_{k,t}, \sigma_k^2) \\
&\propto \frac{q_{i,k}^{\delta_{\{t>1\}}} q_{k,j}^{\delta_{\{t<T\}}}}{\sigma_k} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right). \quad \forall k \in \{1,2,\ldots,K\}. \qquad \text{(B.2)}
\end{aligned}
$$

$Z_{n,t}$ is then sampled from a discrete distribution given by(B.2).

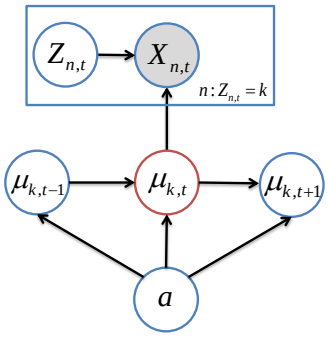

Figure B.3: The local graphical model for $\mu_{k,t}$

B.1.3 Regime Mean $\mu_{k,t}$

From Figure B.3, we derive the full conditional of $\mu_{k,t}$ as follows,

$$
\begin{aligned}
& P(\mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_{n,t}\}_{n=1}^{N}, \sigma_k^2) \\
& \propto P(\mu_{k,t} \mid a, \mu_{k,t-1}) P(\mu_{k,t+1} \mid a, \mu_{k,t}) P(\{X_{n,t}\}_{n=1}^{N} \mid \{Z_{n,t}\}_{n=1}^{N}, \mu_{k,t}, \sigma_k^2) \\
& \propto P(\mu_{k,t} \mid a, \mu_{k,t-1}) P(\mu_{k,t+1} \mid a, \mu_{k,t}) \prod_{n=1}^{N} P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_{n,t}=k\}}}.
\end{aligned}
$$

Note that

1. When $t = 1$, we do not have $P(\mu_{k,t} \mid a, \mu_{k,t-1})$,
2. When $t = T$, we do not have $P(\mu_{k,t+1} \mid a, \mu_{k,t})$,
3. When $t = 1$, we take the stationary distribution of $AR(1)$ as the prior of $\mu_{k,1}$, i.e.,

$$
\mu_{k,1} \sim N\left(m, \frac{\sigma_\varepsilon^2}{1-a^2}\right) \doteq N(m, \sigma_m^2).
$$

Therefore,

$$
\begin{aligned}
& \mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_{n,t}\}_{n=1}^{N}, \sigma_k^2 \\
& \sim N(\mu_{k,t})^{\delta_{\{t=1\}}} N(\mu_{k,t} \mid a\mu_{k,t-1})^{\delta_{\{t>1\}}} N(\mu_{k,t+1} \mid a\mu_{k,t})^{\delta_{\{t<T\}}} \prod_{n=1}^{N} N(x_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_{n,t}=k\}}}.
\end{aligned}
$$

Since it is a normal distribution, we only need to find its mean and variance.

Remark 1 A quick way to get μ and σ^2 for a normal distribution $N(x \mid \mu, \sigma^2)$.

$$-log(N(x \mid \mu, \sigma^2)) = \frac{(x-\mu)^2}{2\sigma^2} + \frac{log(2\pi)}{2} + \frac{log(\sigma^2)}{2} = \theta_2 x^2 - \theta_1 x + \ldots,$$

where $\theta_2 = \frac{1}{2\sigma^2}$ and $\theta_1 = \frac{\mu}{\sigma^2}$. It follows that $\mu = \frac{\theta_1}{2\theta_2}$ and $\sigma^2 = \frac{1}{2\theta_2}$.

$$\begin{aligned}
&- \delta_{\{t=1\}} log(N(\mu_{k,t})) - \delta_{\{t>1\}} log(N(\mu_{k,t} \mid a, \mu_{k,t-1})) \\
&- \delta_{\{t<T\}} log(N(\mu_{k,t+1} \mid a, \mu_{k,t})) - \sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}} log(N(x_{n,t} \mid \mu_{k,t}, \sigma_k^2)) \\
&= \delta_{\{t=1\}} \frac{(\mu_{k,t} - m)^2}{2\sigma_m^2} + \delta_{\{t>1\}} \frac{(\mu_{k,t} - m(1-a) - a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2} \\
&+ \delta_{\{t<T\}} \frac{(\mu_{k,t+1} - m(1-a) - a\mu_{k,t})^2}{2\sigma_\varepsilon^2} + \sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}} \frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2} + \ldots.
\end{aligned}$$

We then have

$$\theta_2 = \frac{\delta_{\{t=1\}}(1-a^2)}{2\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{2\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}} a^2}{2\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}}}{2\sigma_k^2},$$

$$\begin{aligned}
\theta_1 = &\frac{\delta_{\{t=1\}} m(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}} \left(m(1-a) + a\mu_{k,t-1}\right)}{\sigma_\varepsilon^2} \\
&+ \frac{\delta_{\{t<T\}} \left(a\mu_{k,t+1} - ma(1-a)\right)}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}} x_{n,t}}{\sigma_k^2}.
\end{aligned}$$

Therefore, the full conditional for $\mu_{k,t}$ is

$$\mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_{n,t}\}_{n=1}^{N}, \sigma_k^2 \sim N(\mu, \sigma^2), \tag{B.3}$$

where

$$\mu = \frac{\theta_1}{2\theta_2} = \left(\frac{\frac{\delta_{\{t=1\}} m(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}} \left(m(1-a) + a\mu_{k,t-1}\right)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}} \left(a\mu_{k,t+1} - ma(1-a)\right)}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}} x_{n,t}}{\sigma_k^2}}{\frac{\delta_{\{t=1\}}(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}} a^2}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}}}{\sigma_k^2}} \right), \tag{B.4}$$

and

$$\begin{aligned}
\sigma^2 = \frac{1}{2\theta_2} &= \left(\frac{\delta_{\{t=1\}}(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}} a^2}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}}}{\sigma_k^2} \right)^{-1} \\
&= \frac{\sigma_\varepsilon^2 \sigma_k^2}{\left(\delta_{\{t=1\}}(1-a^2) + \delta_{\{t>1\}} + \delta_{\{t<T\}} a^2\right) \sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}}\right) \sigma_\varepsilon^2}.
\end{aligned} \tag{B.5}$$

We rewrite the numerator of μ in (B.4) as

$$\begin{aligned}&\delta_{\{t=1\}} m(1-a^2)\sigma_k^2 + \delta_{\{t>1\}}(m(1-a)+a\mu_{k,t-1})\sigma_k^2 \\ &+ \delta_{\{t<T\}}(a\mu_{k,t+1} - ma(1-a))\sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}} x_{n,t}\right)\sigma_\varepsilon^2,\end{aligned}$$

and the denominator as

$$\left(\delta_{\{t=1\}}(1-a^2) + \delta_{\{t>1\}} + \delta_{\{t<T\}}a^2\right)\sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_{n,t}=k\}}\right)\sigma_\varepsilon^2.$$

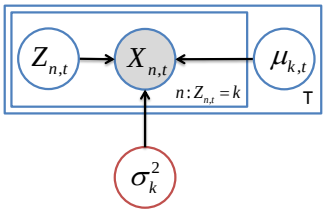

Figure B.4: The local graphical model for σ_k^2

B.1.4 Regime Variance σ_k^2

Since the prior for σ_k is $Unif(0,s)$, where s is the sample standard deviation of $\{X_{n,t}\}$, hence the prior for σ_k^2 is $P(\sigma_k^2) = \frac{1}{2s\sqrt{\sigma_k^2}}$. By Figure B.4, the full conditional for σ_k^2 is

$$
\begin{aligned}
&P(\sigma_k^2 \mid \{\mu_{k,t}\}_{t=1}^T, \{X_{n,t}\}, \{Z_{n,t}\}) \\
&\propto P(\sigma_k^2) P(\{X_{n,t}\} \mid \{Z_{n,t}\}, \{\mu_{k,t}\}_{t=1}^T, \sigma_k^2) \\
&\propto P(\sigma_k^2) \prod_{t=1}^{T} \prod_{n=1}^{N} P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_{n,t}=k\}}} \\
&\propto \frac{1}{\sqrt{\sigma_k^2}} \left(\sigma_k^2\right)^{-\frac{\sum_{t=1}^T \sum_{n=1}^N \delta_{\{Z_{n,t}=k\}}}{2}} exp\left(-\frac{\sum_{t=1}^T \sum_{n=1}^N \delta_{\{Z_{n,t}=k\}} (x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right).
\end{aligned}
$$

Thus,

$$
\sigma_k^2 \mid \{\mu_{k,t}\}_t, \{X_{n,t}\}_{n,t}, \{Z_{n,t}\}_{n,t} \sim inv - Gamma(\alpha_{\sigma_k^2}, \beta_{\sigma_k^2}), \tag{B.6}
$$

where

$$
\text{shape parameter} \quad \alpha_{\sigma_k^2} = \frac{\sum_{t=1}^T \sum_{n=1}^N \delta_{\{Z_{n,t}=k\}} - 1}{2}, \tag{B.7}
$$

$$
\text{scale parameter} \quad \beta_{\sigma_k^2} = \frac{\sum_{t=1}^T \sum_{n=1}^N \delta_{\{Z_{n,t}=k\}} (x_{n,t} - \mu_{k,t})^2}{2}. \tag{B.8}
$$

Since $0 \leq \sigma_k \leq s$, we know $0 \leq \sigma_k^2 \leq s^2$. Samples from (B.6) are rejected if they are larger than s^2, in which case we resample from the prior distribution.

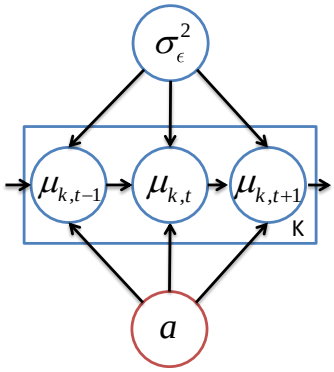

Figure B.5: The local graphical model for a

B.1.5 Regression Coefficient a

Given that the prior for a is $Unif(-1,1)$, the full conditional for a is

$$a \mid \{\mu_{k,t}\} \sim \prod_{k=1}^{K}\prod_{t=2}^{T} N(\mu_{k,t} \mid a, \mu_{k,t-1}),$$

by Figure B.5. Take the logarithm,

$$-\sum_{k=1}^{K}\sum_{t=2}^{T} log(N(\mu_{k,t} \mid a, \mu_{k,t-1})) = \sum_{k=1}^{K}\sum_{t=2}^{T} \frac{((m-\mu_{k,t-1})a-(m-\mu_{k,t}))^2}{2\sigma_\varepsilon^2} + \ldots,$$

we have

$$\theta_2 = \sum_{k=1}^{K}\sum_{t=2}^{T} \frac{(m-\mu_{k,t-1})^2}{2\sigma_\varepsilon^2} \quad \text{and} \quad \theta_1 = \sum_{k=1}^{K}\sum_{t=2}^{T} \frac{(m-\mu_{k,t-1})(m-\mu_{k,t})}{\sigma_\varepsilon^2}.$$

Therefore, the full conditional for a is

$$a \mid \{\mu_{k,t}\} \sim N(\mu_a, \sigma_a^2), \tag{B.9}$$

where

$$\mu_a = \frac{\theta_1}{2\theta_2} = \frac{\sum_{k=1}^{K}\sum_{t=2}^{T}(m-\mu_{k,t-1})(m-\mu_{k,t})}{\sum_{k=1}^{K}\sum_{t=2}^{T}(m-\mu_{k,t-1})^2}, \tag{B.10}$$

$$\sigma_a^2 = \frac{1}{2\theta_2} = \frac{\sigma_\varepsilon^2}{\sum_{k=1}^{K}\sum_{t=2}^{T}(m-\mu_{k,t-1})^2}. \tag{B.11}$$

Since $-1 \le a \le 1$, samples from (B.9) are rejected if they are out of this range, in which case we resample from the prior distribution.

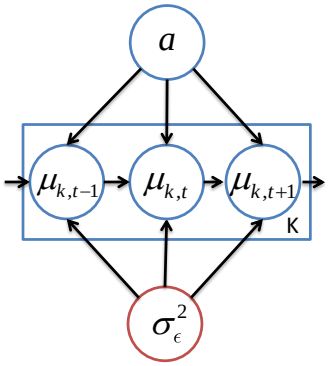

Figure B.6: The local graphical model for a

B.1.6 Regression Variance σ_ε^2

Since the prior for σ_ε is $Unif(0,s)$, the prior for σ_ε^2 is $P(\sigma_\varepsilon^2) = \frac{1}{s\sqrt{\sigma_\varepsilon^2}}$. From Figure B.6, the full conditional for σ_ε^2 is

$$
\begin{aligned}
&P(\sigma_\varepsilon^2 \mid \{\mu_{k,t}\}, a) \propto P(\sigma_\varepsilon^2) \prod_{k=1}^{K} P(\{\mu_{k,t}\}_{t=1}^{T} \mid \sigma_\varepsilon^2, a) \\
&\propto P(\sigma_\varepsilon^2) \prod_{k=1}^{K} \left(P(\mu_{k,1} \mid \sigma_\varepsilon^2, a) \prod_{t=2}^{T} P(\mu_{k,t} \mid \mu_{k,t-1}, \sigma_\varepsilon^2, a) \right) \\
&\propto \frac{1}{\sqrt{\sigma_\varepsilon^2}} \prod_{k=1}^{K} \frac{1}{\sqrt{\sigma_\varepsilon^2}} exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2}{2\sigma_\varepsilon^2} \right) \\
&\quad \times \prod_{k=1}^{K} \prod_{t=2}^{T} \frac{1}{\sqrt{\sigma_\varepsilon^2}} exp\left(-\frac{(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2} \right) \\
&\propto (\sigma_\varepsilon^2)^{-\frac{TK+1}{2}} exp\left(-\frac{\sum_{k=1}^{K}(1-a^2)(\mu_{k,1}-m)^2 + \sum_{k=1}^{K}\sum_{t=2}^{T}(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2} \right)
\end{aligned}
$$

Thus,

$$\sigma_\varepsilon^2 \mid \{\mu_{k,t}\}, a \sim inv-Gamma(\alpha_{\sigma_\varepsilon^2}, \beta_{\sigma_\varepsilon^2}), \tag{B.12}$$

where

$$\text{shape parameter} \quad \alpha_{\sigma_\varepsilon^2} = \frac{TK-1}{2}, \tag{B.13}$$

$$\text{scale parameter} \quad \beta_{\sigma_\varepsilon^2} = \frac{\sum_{k=1}^{K}(1-a^2)(\mu_{k,1}-m)^2 + \sum_{k=1}^{K}\sum_{t=2}^{T}(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2}. \tag{B.14}$$

Since $0 \leq \sigma_\varepsilon \leq s$, we know $0 \leq \sigma_\varepsilon^2 \leq s^2$. Samples from (B.12) are rejected if they are larger than s^2, in which case we resample from the prior distribution.

B.2 Blocked Gibbs Sampling for HMRCA

B.2.1 Blocked Gibbs Sampling for $\mu_{k,t}$

$$\mu_{k,1} \sim N\left(m, \frac{\sigma_\varepsilon^2}{1-a^2}\right).$$

We know that $\mu_{k,t} = m(1-a) + a\mu_{k,t-1} + \varepsilon_k$, $\forall t \in \{2,\ldots,T\}$, where $\varepsilon_k \sim N(0,\sigma_\varepsilon^2)$, and m is the mean of $\{X_{n,t}\}$. Hence,

$$\mu_{k,t} \mid \mu_{k,t-1} \sim N\left(m(1-a) + a\mu_{k,t-1}, \sigma_\varepsilon^2\right).$$

The prior for $\{\mu_{k,t}\}_{t=1}^T$ is therefore

$$\begin{aligned}
P(\{\mu_{k,t}\}_{t=1}^T) &= P(\mu_{k,1})\prod_{t=2}^T P(\mu_{k,t} \mid \mu_{k,t-1}) \\
&\propto \frac{\sqrt{1-a^2}}{\sigma_\varepsilon} exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2}{2\sigma_\varepsilon^2}\right)\prod_{t=2}^T \frac{1}{\sigma_\varepsilon} exp\left(-\frac{(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right) \\
&\propto \frac{\sqrt{1-a^2}}{\sigma_\varepsilon^T} exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2 + \sum_{t=2}^T(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right). \quad \text{(B.15)}
\end{aligned}$$

We want to sample from $P(\{\mu_{k,t}\}_{t=1}^T \mid \{Z_{n,t}\},\{X_{n,t}\},\{\sigma_k^2\})$, $\forall k \in \{1,\ldots,K\}$. The likelihood for $\{\mu_{k,t}\}_{t=1}^T$ is

$$\begin{aligned}
&P(\{X_{n,t}\} \mid \{\mu_{k,t}\}_{t=1}^T, \{Z_{n,t}\}, \sigma_k^2) \\
&\propto \prod_{n=1}^N\prod_{t=1}^T P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_{n,t}=k\}}} \\
&\propto \frac{1}{\sigma_k^{\sum_{n=1}^N\sum_{t=1}^T \delta_{\{Z_{n,t}=k\}}}} exp\left(-\frac{\sum_{n=1}^N\sum_{t=1}^T \delta_{\{Z_{n,t}=k\}}(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2}\right). \quad \text{(B.16)}
\end{aligned}$$

Based on (B.30) and (B.31), $\forall k \in \{1,\ldots,K\}$,

$$P(\{\mu_{k,t}\}_{t=1}^T \mid \{Z_{n,t}\},\{X_{n,t}\},\{\sigma_k^2\}) \propto P(\{\mu_{k,t}\}_{t=1}^T)P(\{X_{n,t}\} \mid \{\mu_{k,t}\}_{t=1}^T, \{Z_{n,t}\}, \sigma_k^2), \quad \text{(B.17)}$$

which is a multivariate normal distribution.

Remark 2 To calculate the kernel of Gaussian: $(\mu_k - c)^T \Sigma^{-1} (\mu_k - c)$ from a quadratic function of $\{\mu_{k,t}\}_{t=1}^T$, where $\mu_k = [\mu_{k,1}, \ldots, \mu_{k,T}]^T$, we first transform the function into $f^T \mu_k + \mu_k^T H \mu_k + \ldots$, and then compare the coefficients: $H = \Sigma^{-1}$ and $f = -2\Sigma^{-1} c$. It follows that

$$\Sigma = H^{-1} \qquad \text{and} \qquad c = -H^{-1} f/2. \tag{*}$$

We now focus on the quadratic and linear terms of $\{\mu_{k,t}\}$ in the exponential kernel of Gaussian. We ignore σ_ε^2 or σ_k^2 in front of the exponential kernel, because $\{\mu_{k,t}^2\}$ and $\{\mu_{k,t}\}$ uniquely identify the multivariate normal.

In (B.30),

$$\begin{aligned}
&exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2 + \sum_{t=2}^T (\mu_{k,t} - m(1-a) - a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right) \\
&\propto exp\left(-\frac{1}{2\sigma_\varepsilon^2}\Big(\big(\mu_{k,1}^2 + (1+a^2)\sum_{t=2}^{T-1}\mu_{k,t}^2 + \mu_{k,T}^2\big) - 2a\sum_{t=1}^{T-1}\mu_{k,t}\mu_{k,t+1}\Big)\right) \\
&\quad \times exp\left(\frac{1}{2\sigma_\varepsilon^2}\Big(2m(1-a)\big(\mu_{k,1} + (1-a)\sum_{t=2}^{T-1}\mu_{k,t} + \mu_{k,T}\big)\Big)\right) \\
&\propto exp\left(-\frac{1}{2}(f_1^T \mu_k + \mu_k^T H_1 \mu_k)\right),
\end{aligned} \tag{B.18}$$

where f_1^T is a row vector with T elements, and H_1 is a $T \times T$ matrix

$$f_1^T = -\frac{2m(1-a)}{\sigma_\varepsilon^2}[1, (1-a), \ldots, (1-a), 1],$$

$$H_1 = \frac{1}{\sigma_\varepsilon^2}\begin{bmatrix}
1 & -a & 0 & \ldots & 0 & 0 \\
-a & 1+a^2 & -a & \ldots & 0 & 0 \\
0 & -a & 1+a^2 & \ldots & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & -a & 1+a^2 & -a \\
0 & 0 & \ldots & 0 & -a & 1
\end{bmatrix}.$$

In (B.31),

$$
\begin{aligned}
& exp\left(-\frac{\sum_{n=1}^{N}\sum_{t=1}^{T}\delta_{\{Z_{n,t}=k\}}(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2}\right) \\
& \propto exp\left(-\frac{1}{2\sigma_k^2}\sum_{t=1}^{T}\Big(\sum_{n=1}^{N}\delta_{\{Z_{n,t}=k\}}\mu_{k,t}^2-2\sum_{n=1}^{N}\delta_{\{Z_{n,t}=k\}}x_{n,t}\mu_{k,t}\Big)\right) \\
& \propto exp\left(-\frac{1}{2}(f_2^T\mu_k+\mu_k^T H_2\mu_k)\right), \qquad \text{(B.19)}
\end{aligned}
$$

where f_1^T is a row vector with T elements and H_1 is a $T\times T$ matrix

$$
f_2^T=-\frac{2}{\sigma_k^2}[\sum_{n=1}^{N}\delta_{\{Z_{n,1}=k\}}x_{n,1},\ldots,\sum_{n=1}^{N}\delta_{\{Z_{n,T}=k\}}x_{n,T}],
$$

$$
H_2=\frac{1}{\sigma_k^2}\begin{bmatrix}
\sum_{n=1}^{N}\delta_{\{Z_{n,1}=k\}} & 0 & \ldots & 0 & 0 \\
0 & \sum_{n=1}^{N}\delta_{\{Z_{n,2}=k\}} & \ldots & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & \sum_{n=1}^{N}\delta_{\{Z_{n,T-1}=k\}} & 0 \\
0 & 0 & \ldots & 0 & \sum_{n=1}^{N}\delta_{\{Z_{n,T}=k\}}
\end{bmatrix}.
$$

Based on (B.18) and (B.19), f and H in $P(\{\mu_{k,t}\}_{t=1}^{T}\mid\{Z_{n,t}\},\{X_{n,t}\},\{\sigma_k^2\})$ are

$$
f=f_1+f_2,\quad H=H_1+H_2.
$$

Therefore,

$$
\{\mu_{k,t}\}_{t=1}^{T}\mid\{Z_{n,t}\},\{X_{n,t}\},\{\sigma_k^2\}\sim N(c,\Sigma), \qquad \text{(B.20)}
$$

where c is a vector with T elements, and Σ is a $T\times T$ matrix

$$
c=-(H_1+H_2)^{-1}(f_1+f_2)/2,\quad \Sigma=(H_1+H_2)^{-1}.
$$

B.2.2 Blocked Gibbs Sampling for $Z_{n,t}$

Support that there are K regimes,

- $\{\pi_k\}_{k=1}^K$ is their discrete prior distribution when $t = 1$,
- $b_k(X_{n,t})$ is the likelihood for $X_{n,t}$ if it belongs to the k^{th} regime,
- $Q = \{q_{ij}\}$ is the transition matrix for the regimes,
- $\{\mu_{k,t}\}_{t=1}^T$ is the mean for the k^{th} regime.

Let $\alpha_k(t) = P(\{X_{n,t'}\}_{t'=1}^t, Z_{n,t} = k \mid \{\mu_{k,t}\}, Q)$ and $\beta_k(t) = P(\{X_{n,t'}\}_{t'=t+1}^T \mid Z_{n,t} = k, \{\mu_{k,t}\}, Q)$. The forward and backward algorithms are:

1. $\alpha_k(1) = \pi_k b_k(X_{n,1})$,
2. $\alpha_k(t+1) = \left[\sum_{i=1}^K \alpha_i(t) q_{i,k}\right] b_k(X_{n,t+1})$, $\forall t \in \{2,\ldots,T\}$,
3. $\beta_k(T) = 1$,
4. $\beta_k(t) = \sum_{j=1}^K q_{k,j} b_j(X_{n,t+1}) \beta_j(t+1)$. $\forall t \in \{1,\ldots,T-1\}$.

We focus on one server n to update $\{Z_{n,t}\}$, since the $\{Z_{n,t}\}$ are independent across n, conditional on the other parameters of the model.

For $\{Z_{n,1}\}$,

$$
\begin{aligned}
&P(Z_{n,1} = k \mid \{X_{n,t'}\}_{t'=1}^T, \{\mu_{k,t}\}, Q) \\
&\propto P(Z_{n,1} = k, \{X_{n,t'}\}_{t'=1}^T \mid \{\mu_{k,t}\}, Q) \\
&\propto P(X_{n,1}, Z_{n,1} = k \mid \{\mu_{k,t}\}, Q) P(\{X_{n,t'}\}_{t'=2}^T \mid Z_{n,1} = k, \{\mu_{k,t}\}, Q) \\
&\propto \frac{\pi_k}{\sigma_k} exp\Big(-\frac{(x_{n,1} - \mu_{k,1})^2}{2\sigma_k^2}\Big) \beta_k(1), \quad \forall k \in \{1,\ldots,K\}.
\end{aligned} \tag{B.21}
$$

For $\{Z_{n,t}\}$, $\forall t \in \{2,\ldots,T\}$,

$$
\begin{aligned}
&P(Z_{n,t}=k \mid Z_{n,t-1}=i, \{X_{n,t'}\}_{t'=1}^{T}, \{\mu_{k,t}\}, Q) \\
&\propto P(Z_{n,t}=k \mid Z_{n,t-1}=i, \{X_{n,t'}\}_{t'=t}^{T}, \{\mu_{k,t}\}, Q) \\
&\propto P(Z_{n,t}=k \mid Z_{n,t-1}=i) P(X_{n,t} \mid Z_{n,t}=k, \{\mu_{k,t}\}, Q) P(\{X_{n,t'}\}_{t'=t+1}^{T} \mid Z_{n,t}=k, \{\mu_{k,t}\}, Q) \\
&\propto \frac{q_{i,k}}{\sigma_k} exp\big(-\frac{(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2}\big)\, \beta_k(t), \quad \forall k \in \{1,\ldots,K\}.
\end{aligned} \tag{B.22}
$$

We sample $Z_{n,1}$ from (B.21), and sample $\{Z_{n,t}\}_{t=2}^{T}$ subsequently from (B.22). It is equivalent that we sample from the joint distribution

$$P(\{Z_{n,t'}\}_{t'=1}^{T} \mid \{X_{n,t'}\}_{t'=1}^{T}, \{\mu_{k,t}\}, Q).$$

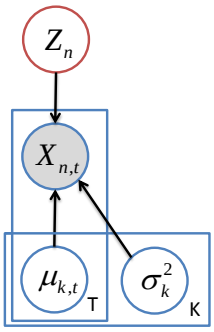

Figure B.7: The local graphical model for Z_n

B.3 Gibbs Sampling for HMA

a and σ_ε^2 are updated according to (B.9) and (B.12) as in Section B.1. It suffices to adjust the updating formula of $\{Z_n\}, \{\mu_{k,t}\}, \{\sigma_k^2\}$.

B.3.1 Regime Indicator Z_n

Note that Z_n has a discrete prior distribution $\{\pi_k\}$ on $[1,\ldots,K]$, where $\{\pi_k\}$ follows a dirichlet distribution with $\alpha = 1$ for each π_k. We update Z_n for $n \in \{1,2,\ldots,N\}$. From Figure B.7 we have

$$\begin{aligned} & P(Z_n = k \mid \{x_{n,t}\}_{t=1}^T, \{\mu_{k,t}\}, \sigma_k^2) \\ & \propto P(Z_n = k) \prod_{t=1}^T P(X_{n,t} \mid Z_n = k, \mu_{k,t}, \sigma_k^2) \\ & \propto \pi_k \prod_{t=1}^T \frac{1}{\sigma_k} exp\left(-\frac{(x_{n,t} - \mu_{k,t})^2}{2\sigma_k^2}\right). \quad \forall k \in \{1,2,\ldots,K\}. \end{aligned} \tag{B.23}$$

Z_n is then sampled from a discrete distribution given by (B.23).

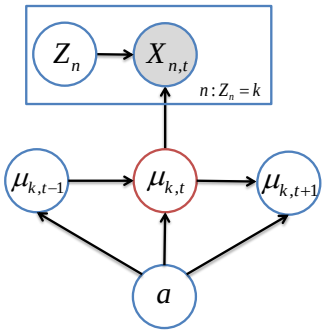

Figure B.8: The local graphical model for $\mu_{k,t}$

B.3.2 Regime Mean $\mu_{k,t}$

From Figure B.8, we derive the full conditional of $\mu_{k,t}$ as follows,

$$
\begin{aligned}
&P(\mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_n\}, \sigma_k^2) \\
&\propto P(\mu_{k,t} \mid a, \mu_{k,t-1}) P(\mu_{k,t+1} \mid a, \mu_{k,t}) P(\{X_{n,t}\}_{n=1}^{N} \mid \{Z_n\}, \mu_{k,t}, \sigma_k^2) \\
&\propto P(\mu_{k,t} \mid a, \mu_{k,t-1}) P(\mu_{k,t+1} \mid a, \mu_{k,t}) \prod_{n=1}^{N} P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_n=k\}}}.
\end{aligned}
$$

Note that

1. When $t = 1$, we do not have $P(\mu_{k,t} \mid a, \mu_{k,t-1})$.
2. When $t = T$, we do not have $P(\mu_{k,t+1} \mid a, \mu_{k,t})$.
3. When $t = 1$, we take the stationary distribution of $AR(1)$ as the prior of $\mu_{k,1}$, i.e.,

$$
\mu_{k,1} \sim N\left(m, \frac{\sigma_\varepsilon^2}{1-a^2}\right) \doteq N(m, \sigma_m^2).
$$

Therefore,

$$
\begin{aligned}
&\mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_n\}, \sigma_k^2 \\
&\sim N(\mu_{k,t})^{\delta_{\{t=1\}}} N(\mu_{k,t} \mid a\mu_{k,t-1})^{\delta_{\{t>1\}}} N(\mu_{k,t+1} \mid a\mu_{k,t})^{\delta_{\{t<T\}}} \prod_{n=1}^{N} N(x_{n,t} \mid \mu_{k,t}, \sigma_k^2)^{\delta_{\{Z_n=k\}}}.
\end{aligned}
$$

Since it is a normal distribution, we only need to find its mean and variance.

According to Remark B.1.3,

$$
\begin{aligned}
&-\delta_{\{t=1\}} log(N(\mu_{k,t})) - \delta_{\{t>1\}} log(N(\mu_{k,t} \mid a, \mu_{k,t-1})) \\
&-\delta_{\{t<T\}} log(N(\mu_{k,t+1} \mid a, \mu_{k,t})) - \sum_{n=1}^{N} \delta_{\{Z_n=k\}} log(N(x_{n,t} \mid \mu_{k,t}, \sigma_k^2)) \\
&= \delta_{\{t=1\}} \frac{(\mu_{k,t}-m)^2}{2\sigma_m^2} + \delta_{\{t>1\}} \frac{(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2} \\
&+ \delta_{\{t<T\}} \frac{(\mu_{k,t+1}-m(1-a)-a\mu_{k,t})^2}{2\sigma_\varepsilon^2} + \sum_{n=1}^{N} \delta_{\{Z_n=k\}} \frac{(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2} + \ldots.
\end{aligned}
$$

We then have

$$
\begin{aligned}
\theta_2 &= \frac{\delta_{\{t=1\}}(1-a^2)}{2\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{2\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}}a^2}{2\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}}}{2\sigma_k^2}, \\
\theta_1 &= \frac{\delta_{\{t=1\}}m(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}\left(m(1-a)+a\mu_{k,t-1}\right)}{\sigma_\varepsilon^2} \\
&\quad + \frac{\delta_{\{t<T\}}\left(a\mu_{k,t+1}-ma(1-a)\right)}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}} x_{n,t}}{\sigma_k^2}.
\end{aligned}
$$

Therefore, the full conditional for $\mu_{k,t}$ is

$$
\mu_{k,t} \mid a, \mu_{k,t-1}, \mu_{k,t+1}, \{X_{n,t}\}_{n=1}^{N}, \{Z_n\}, \sigma_k^2 \sim N(\mu, \sigma^2), \tag{B.24}
$$

where

$$
\mu = \frac{\theta_1}{2\theta_2} = \left(\frac{\frac{\delta_{\{t=1\}}m(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}\left(m(1-a)+a\mu_{k,t-1}\right)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}}\left(a\mu_{k,t+1}-ma(1-a)\right)}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}} x_{n,t}}{\sigma_k^2}}{\frac{\delta_{\{t=1\}}(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}}a^2}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}}}{\sigma_k^2}} \right). \tag{B.25}
$$

$$
\begin{aligned}
\sigma^2 = \frac{1}{2\theta_2} &= \left(\frac{\delta_{\{t=1\}}(1-a^2)}{\sigma_\varepsilon^2} + \frac{\delta_{\{t>1\}}}{\sigma_\varepsilon^2} + \frac{\delta_{\{t<T\}}a^2}{\sigma_\varepsilon^2} + \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}}}{\sigma_k^2} \right)^{-1} \\
&= \frac{\sigma_\varepsilon^2 \sigma_k^2}{\left(\delta_{\{t=1\}}(1-a^2) + \delta_{\{t>1\}} + \delta_{\{t<T\}}a^2\right)\sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_n=k\}}\right)\sigma_\varepsilon^2}.
\end{aligned} \tag{B.26}
$$

We rewrite the numerator of μ in (B.25) as

$$\begin{aligned}&\delta_{\{t=1\}} m(1-a^2)\sigma_k^2 + +\delta_{\{t>1\}}(m(1-a)+a\mu_{k,t-1})\sigma_k^2 \\ &+ \delta_{\{t<T\}}(a\mu_{k,t+1} - ma(1-a))\sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_n=k\}} x_{n,t}\right)\sigma_\varepsilon^2,\end{aligned}$$

and the denominator as

$$\left(\delta_{\{t=1\}}(1-a^2) + \delta_{\{t>1\}} + \delta_{\{t<T\}}a^2\right)\sigma_k^2 + \left(\sum_{n=1}^{N} \delta_{\{Z_n=k\}}\right)\sigma_\varepsilon^2.$$

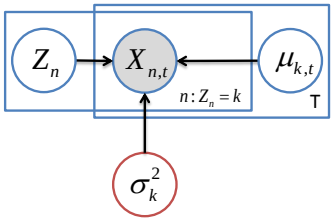

Figure B.9: The local graphical model for σ_k^2

B.3.3 Regime Variance σ_k^2

Since the prior for σ_k is $Unif(0,s)$, where s is the sample standard deviation of $\{X_{n,t}\}$, hence the prior for σ_k^2 is $P(\sigma_k^2) = \frac{1}{2s\sqrt{\sigma_k^2}}$. By Figure B.9, the full conditional for σ_k^2 is

$$
\begin{aligned}
&P(\sigma_k^2 \mid \{\mu_{k,t}\}_{t=1}^T, \{X_{n,t}\}, \{Z_n\}) \\
&\propto P(\sigma_k^2) P(\{X_{n,t}\} \mid \{Z_n\}, \{\mu_{k,t}\}_{t=1}^T, \sigma_k^2) \\
&\propto P(\sigma_k^2) \prod_{n=1}^{N} \left(\prod_{t=1}^{T} P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2) \right)^{\delta_{\{Z_n=k\}}} \\
&\propto \frac{1}{\sqrt{\sigma_k^2}} \left(\sigma_k^2\right)^{-\frac{T\sum_{n=1}^{N} \delta_{\{Z_n=k\}}}{2}} exp\left(-\frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}} \left(\sum_{t=1}^{T} (x_{n,t} - \mu_{k,t})^2 \right)}{2\sigma_k^2} \right).
\end{aligned}
$$

Thus,

$$
\sigma_k^2 \mid \{\mu_{k,t}\}_{t=1}^T, \{X_{n,t}\}, \{Z_n\} \sim inv - Gamma(\alpha_{\sigma_k^2}, \beta_{\sigma_k^2}), \tag{B.27}
$$

where

$$
\text{shape parameter} \quad \alpha_{\sigma_k^2} = \frac{T\sum_{n=1}^{N} \delta_{\{Z_n=k\}} - 1}{2}, \tag{B.28}
$$

$$
\text{scale parameter} \quad \beta_{\sigma_k^2} = \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}} \left(\sum_{t=1}^{T} (x_{n,t} - \mu_{k,t})^2 \right)}{2}. \tag{B.29}
$$

Since $0 \leq \sigma_k \leq s$, we know $0 \leq \sigma_k^2 \leq s^2$. Samples from (B.27) are rejected if they are larger than s^2, in which case we resample from the prior distribution.

B.4 Blocked Gibbs Sampling for HMA

B.4.1 Blocked Gibbs Sampling for $\mu_{k,t}$

$$\mu_{k,1} \sim N\left(m, \frac{\sigma_\varepsilon^2}{1-a^2}\right).$$

We know that $\mu_{k,t} = m(1-a) + a\mu_{k,t-1} + \varepsilon_k$, $\forall t \in \{2,\ldots,T\}$, where $\varepsilon_k \sim N(0,\sigma_\varepsilon^2)$, and m is the mean of $\{X_{n,t}\}$. Hence,

$$\mu_{k,t} \mid \mu_{k,t-1} \sim N\left(m(1-a) + a\mu_{k,t-1}, \sigma_\varepsilon^2\right).$$

The prior for $\{\mu_{k,t}\}_{t=1}^T$ is therefore

$$\begin{aligned}
&P(\{\mu_{k,t}\}_{t=1}^T) = P(\mu_{k,1})\prod_{t=2}^T P(\mu_{k,t} \mid \mu_{k,t-1}) \\
&\propto \frac{\sqrt{1-a^2}}{\sigma_\varepsilon} exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2}{2\sigma_\varepsilon^2}\right)\prod_{t=2}^T \frac{1}{\sigma_\varepsilon} exp\left(-\frac{(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right) \\
&\propto \frac{\sqrt{1-a^2}}{\sigma_\varepsilon^T} exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2 + \sum_{t=2}^T(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right). \quad \text{(B.30)}
\end{aligned}$$

We want to sample from $P(\{\mu_{k,t}\}_{t=1}^T \mid \{Z_n\},\{X_{n,t}\},\{\sigma_k^2\})$, $\forall k \in \{1,\ldots,K\}$. The likelihood for $\{\mu_{k,t}\}_{t=1}^T$ is

$$\begin{aligned}
&P(\{X_{n,t}\} \mid \{\mu_{k,t}\}_{t=1}^T, \{Z_n\}, \sigma_k^2) \\
&\propto \prod_{n=1}^N \Big(\prod_{t=1}^T P(X_{n,t} \mid \mu_{k,t}, \sigma_k^2)\Big)^{\delta_{\{Z_n=k\}}} \\
&\propto \frac{1}{\sigma_k^{T\sum_{n=1}^N \delta_{\{Z_n=k\}}}} exp\left(-\frac{\sum_{n=1}^N\sum_{t=1}^T \delta_{\{Z_n=k\}}(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2}\right). \quad \text{(B.31)}
\end{aligned}$$

Based on (B.30) and (B.31), $\forall k \in \{1,\ldots,K\}$,

$$P(\{\mu_{k,t}\}_{t=1}^T \mid \{Z_n\},\{X_{n,t}\},\{\sigma_k^2\}) \propto P(\{\mu_{k,t}\}_{t=1}^T)P(\{X_{n,t}\} \mid \{\mu_{k,t}\}_{t=1}^T, \{Z_n\}, \sigma_k^2), \quad \text{(B.32)}$$

which is a multivariate normal distribution.

According to Remark B.2.1, we now focus on the quadratic and linear terms of $\{\mu_{k,t}\}$ in the exponential kernel of Gaussian. We ignore σ_ε^2 or σ_k^2 in front of the exponential kernel, because $\{\mu_{k,t}^2\}$ and $\{\mu_{k,t}\}$ uniquely identify the multivariate normal.

In (B.30),

$$
\begin{aligned}
&exp\left(-\frac{(1-a^2)(\mu_{k,1}-m)^2+\sum_{t=2}^{T}(\mu_{k,t}-m(1-a)-a\mu_{k,t-1})^2}{2\sigma_\varepsilon^2}\right)\\
&\propto exp\left(-\frac{1}{2\sigma_\varepsilon^2}\Big(\big(\mu_{k,1}^2+(1+a^2)\sum_{t=2}^{T-1}\mu_{k,t}^2+\mu_{k,T}^2\big)-2a\sum_{t=1}^{T-1}\mu_{k,t}\mu_{k,t+1}\Big)\right) \qquad \text{(B.33)}\\
&\quad\times exp\left(\frac{1}{2\sigma_\varepsilon^2}\Big(2m(1-a)\big(\mu_{k,1}+(1-a)\sum_{t=2}^{T-1}\mu_{k,t}+\mu_{k,T}\big)\Big)\right)\\
&\propto exp\left(-\frac{1}{2}(f_1^T\mu_k+\mu_k^T H_1\mu_k)\right), \qquad \text{(B.34)}
\end{aligned}
$$

where f_1^T is a row vector with T elements and H_1 is a $T\times T$ matrix

$$
f_1^T=-\frac{2m(1-a)}{\sigma_\varepsilon^2}[1,(1-a),\ldots,(1-a),1],
$$

$$
H_1=\frac{1}{\sigma_\varepsilon^2}\begin{bmatrix}
1 & -a & 0 & \ldots & 0 & 0\\
-a & 1+a^2 & -a & \ldots & 0 & 0\\
0 & -a & 1+a^2 & \ldots & 0 & 0\\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots\\
0 & 0 & \ldots & -a & 1+a^2 & -a\\
0 & 0 & \ldots & 0 & -a & 1
\end{bmatrix}.
$$

In (B.31),

$$
\begin{aligned}
&exp\left(-\frac{\sum_{n=1}^{N}\sum_{t=1}^{T}\delta_{\{Z_n=k\}}(x_{n,t}-\mu_{k,t})^2}{2\sigma_k^2}\right)\\
&\propto exp\left(-\frac{1}{2\sigma_k^2}\sum_{t=1}^{T}\Big(\sum_{n=1}^{N}\delta_{\{Z_n=k\}}\mu_{k,t}^2-2\sum_{n=1}^{N}\delta_{\{Z_n=k\}}x_{n,t}\mu_{k,t}\Big)\right)\\
&\propto exp\left(-\frac{1}{2}(f_2^T\mu_k+\mu_k^T H_2\mu_k)\right) \qquad \text{(B.35)}
\end{aligned}
$$

where f_1^T is a row vector with T elements and H_1 is a $T \times T$ matrix

$$f_2^T = -\frac{2}{\sigma_k^2}[\sum_{n=1}^{N} \delta_{\{Z_n=k\}} x_{n,1}, \ldots, \sum_{n=1}^{N} \delta_{\{Z_n=k\}} x_{n,T}],$$

$$H_2 = \frac{\sum_{n=1}^{N} \delta_{\{Z_n=k\}}}{\sigma_k^2} \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

Based on (B.34) and (B.35), f and H in $P(\{\mu_{k,t}\}_{t=1}^T \mid \{Z_n\}, \{X_{n,t}\}, \{\sigma_k^2\})$ are

$$f = f_1 + f_2, \quad H = H_1 + H_2.$$

Therefore,

$$\{\mu_{k,t}\}_{t=1}^T \mid \{Z_n\}, \{X_{n,t}\}, \{\sigma_k^2\} \sim N(c, \Sigma) \tag{B.36}$$

where c is a vector with T elements, and Σ is a $T \times T$ matrix

$$c = -(H_1 + H_2)^{-1}(f_1 + f_2)/2, \quad \Sigma = (H_1 + H_2)^{-1}.$$

B.5 Metropolis-within-Gibbs for HSMRCA

B.5.1 Transition Matrix Q

Similar to the sampling in HMRCA, we set $n_k = \#\{Z^*_{n,l} : Z^*_{n,l-1} = i, Z^*_{n,l} = k,\ \forall n \in \{1,2,\ldots,N\}\}$, and $Q_i = (q_{i,1},\ldots,q_{i,K})$, i.e. $Q = (Q_1,\ldots,Q_K)^T$. Suppose the prior for Q_i is $Q_i \sim Dir(\alpha_i)$, the posterior distributions of $Q_i \mid \{Z^*_{n,l}\}$ is also a Dirichlet distribution.

$$Q_i \mid \{Z^*_{n,l}\} \sim Dir(\alpha_{i,1}+n_1,\ldots,\alpha_{i,K}+n_K). \quad \forall i \in \{1,2,\ldots,K\}. \tag{B.37}$$

B.5.2 Segment Indicator $Z^*_{n,l}$

We update $Z^*_{n,l}$ for $n \in \{1,2,\ldots,N\}$ and $t_{n,l} \in \{1,2,\ldots,T\}$, where we assume $Z^*_{n,l-1} = i$ and $Z^*_{n,l+1} = j$. The conditional probability is

$$\begin{aligned}
&P(Z^*_{n,l} = k \mid Z^*_{n,l-1} = i, Z^*_{n,l+1} = j, x_{n,t}, \{\mu_{k,t}\}, \{\sigma^2_k\}, Q) \\
&\propto P(Z^*_{n,l} = k \mid Z^*_{n,l-1} = i, Q)^{\delta_{\{t_{n,l}>1\}}} \times P(Z^*_{n,l+1} = j \mid Z^*_{n,l} = k, Q)^{\delta_{\{t_{n,l}<T\}}} \\
&\quad \times \prod_{t=t_{n,l}}^{t_{n,l+1}-1} P(X_{n,t} \mid Z_{n,t} = k, \mu_{k,t}, \sigma^2_k) \\
&\propto q_{i,k}^{\delta_{\{t_{n,l}>1\}}} q_{k,j}^{\delta_{\{t_{n,l}<T\}}} \prod_{t=t_{n,l}}^{t_{n,l+1}-1} exp\left(-\frac{(x_{n,t}-\mu_{k,t})^2}{2\sigma^2_k}\right)/\sigma_k. \quad \forall k \in \{1,2,\ldots,K\}.
\end{aligned} \tag{B.38}$$

$Z^*_{n,l}$ then is sampled from a discrete distribution given by (B.2).

B.5.3 Sojourn Time Mean λ

Suppose the prior for λ is $\lambda \sim Gamma(\alpha, \beta)$, the posterior probability is

$$\begin{aligned} P(\lambda \mid \{\tau_{n,l}\}) &\propto P(\lambda) \prod_{n=1}^{N} \left(\prod_{l=1}^{L_n} P(\tau_{n,l} \mid \lambda) \right) \\ &\propto \lambda^{\alpha-1} exp(-\beta\lambda) \prod_{n=1}^{N} \left(\prod_{l=1}^{L_n} \lambda^{\tau_{n,l}} exp(-\lambda) \right) \\ &\propto \lambda^{(\alpha+\sum_{n=1}^{N}\sum_{l=1}^{L_n}\tau_{n,l})-1} exp\left(-(\sum_{n=1}^{N} L_n + \beta)\lambda \right). \end{aligned}$$

Therefore,

$$\lambda \sim Gamma(\alpha + \sum_{n=1}^{N}\sum_{l=1}^{L_n}\tau_{n,l}, \sum_{n=1}^{N} L_n + \beta). \tag{B.39}$$

APPENDIX C

APPENDIX OF CHAPTER 6

C.1 Generative Methods: Binary Classification in "Click Bot"

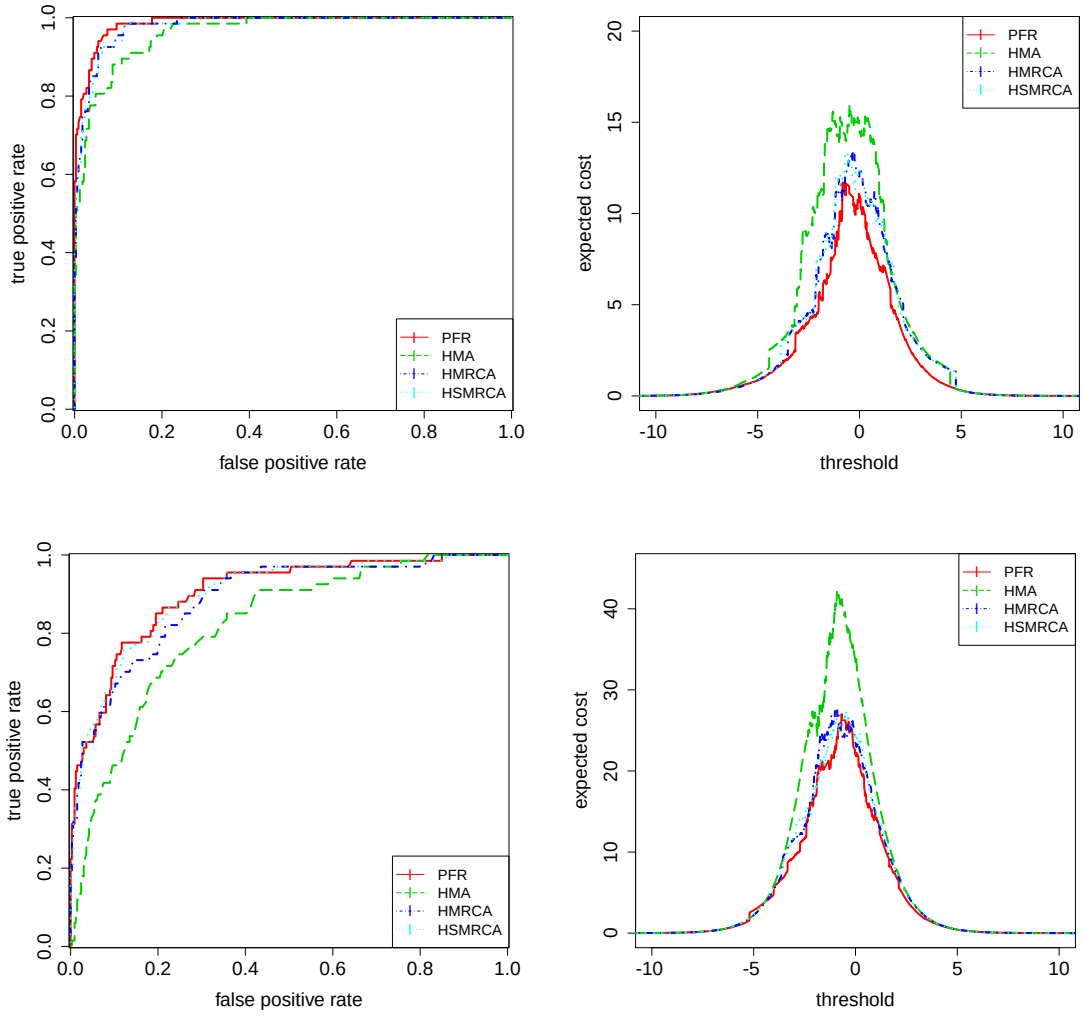

Figure C.1: Generative methods: ROC Curve (upper left) and Cost Curve (upper right) from "Click Bot" Data in processor metric, ROC Curve (lower left) and Cost Curve (lower right) from "Click Bot" Data in memory metric

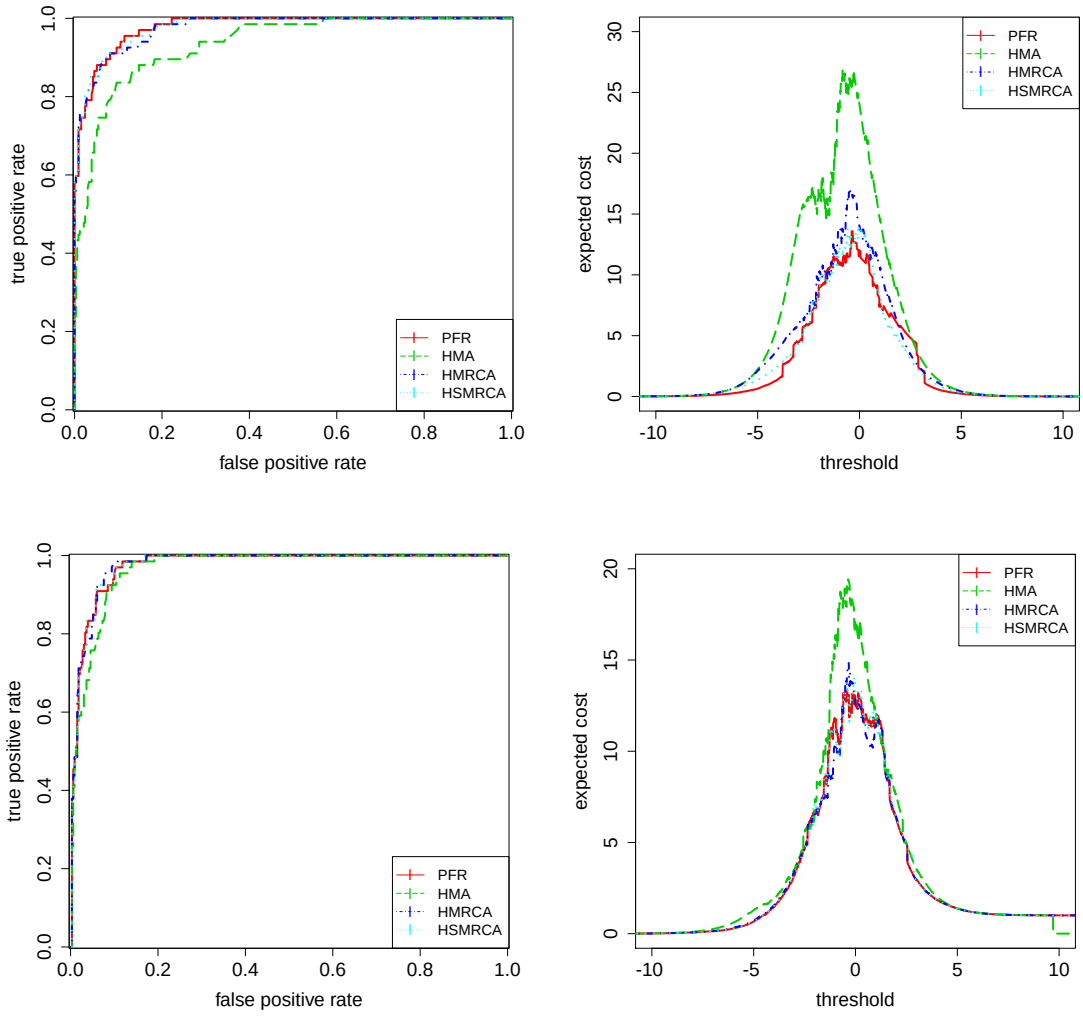

Figure C.2: Generative methods: ROC Curve (upper left) and Cost Curve (upper right) from "Click Bot" Data in disk metric, ROC Curve (lower left) and Cost Curve (lower right) from "Click Bot" Data in network metric

BIBLIOGRAPHY

Absil, P-A, Christopher G Baker, Kyle A Gallivan. 2007. Trust-region methods on riemannian manifolds. *Foundations of Computational Mathematics* **7**(3) 303–330.

Absil, P-A, Robert Mahony, Rodolphe Sepulchre. 2009. *Optimization algorithms on matrix manifolds*. Princeton University Press.

Adler, Roy L, Jean-Pierre Dedieu, Joseph Y Margulies, Marco Martens, Mike Shub. 2002. Newton's method on riemannian manifolds and a geometric model for the human spine. *IMA Journal of Numerical Analysis* **22**(3) 359–390.

Admanti, AR, Paul Pfleiderer. 1991. Sunshine trading and financial market equilibrium. *Review of Financial Studies* **4**(3) 443–481.

Aja-Fernández, Santiago. 2009. *Tensors in image processing and computer vision*. Springer.

Albert, James H, Siddhartha Chib. 1993. Bayes inference via gibbs sampling of autoregressive time series subject to markov mean and variance shifts. *Journal of Business & Economic Statistics* **11**(1) 1–15.

Alter, Orly, Patrick O Brown, David Botstein. 2000. Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences* **97**(18) 10101–10106.

Amini, Arash A, Martin J Wainwright. 2008. High-dimensional analysis of semidefinite relaxations for sparse principal components. *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*. IEEE, 2454–2458.

Andersen, Torben G, Tim Bollerslev, Peter F Christoffersen, Francis X Diebold. 2006. Volatility and correlation forecasting. *Handbook of economic forecasting* **1** 777–878.

Andrieu, Christophe, Arnaud Doucet. 1999. Joint bayesian model selection and estimation of noisy sinusoids via reversible jump mcmc. *Signal Processing, IEEE Transactions on* **47**(10) 2667–2676.

Asteris, Megasthenis, Dimitris S Papailiopoulos, George N Karystinos. 2011. Sparse principal component of a rank-deficient matrix. *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*. IEEE, 673–677.

Avellaneda, Marco, Jeong-Hyun Lee. 2010. Statistical arbitrage in the us equities market. *Quantitative Finance* **10**(7) 761–782.

Baker, Christopher G, P-A Absil, Kyle A Gallivan. 2008. An implicit trust-region method on riemannian manifolds. *IMA journal of numerical analysis* **28**(4) 665–689.

Barclay, Michael J, Jerold B Warner. 1993. Stealth trading and volatility: Which trades move prices? *Journal of Financial Economics* **34**(3) 281–305.

Bertsimas, Dimitris, David B Brown, Constantine Caramanis. 2011. Theory and applications of robust optimization. *SIAM review* **53**(3) 464–501.

Bertsimas, Dimitris, Rahul Mazumder. 2013. Least quantile of squares regression via modern optimization. *arXiv preprint arXiv:1310.8625* .

Bertsimas, Dimitris, Santosh Vempala. 2004. Solving convex programs by random walks. *Journal of the ACM (JACM)* **51**(4) 540–556.

Bishop, Christopher M, et al. 2006. *Pattern recognition and machine learning*, vol. 1. springer New York.

Bollerslev, Tim, Hans Ole Mikkelsen. 1996. Modeling and pricing long memory in stock market volatility. *Journal of Econometrics* **73**(1) 151–184.

Bookstaber, Richard. 2000. Understanding and monitoring the liquidity crisis cycle. *Financial Analysts Journal* 17–22.

Boss, Michael, Helmut Elsinger, Martin Summer, Stefan Thurner 4. 2004. Network topology of the interbank market. *Quantitative Finance* **4**(6) 677–684.

Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends{\textregistered} in Machine Learning* **3**(1) 1–122.

Buntine, Wray. 2002. Variational extensions to em and multinomial pca. *Machine Learning: ECML 2002*. Springer, 23–34.

Cadima, Jorge, Ian T Jolliffe. 1995. Loading and correlations in the interpretation of principle compenents. *Journal of Applied Statistics* **22**(2) 203–214.

Cai, Jian-Feng, Emmanuel J Candès, Zuowei Shen. 2010. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization* **20**(4) 1956–1982.

Candes, Emmanuel, Terence Tao. 2007. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics* 2313–2351.

Candès, Emmanuel J, Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* **9**(6) 717–772.

Cardot, H, F Ferraty, P Sarda. 2003. Spline estimators for the functional linear model. *Statistica Sinica* **13**(3) 571–592.

Cardot, H, P Sarda. 2005. Estimation in generalized linear models for functional data via penalized likelihood. *Journal of Multivariate Analysis* **92**(1) 24–41.

Cardota, H Ã, F Ã Ã Ferratyb, P Sardab. 1999. Functional linear model. *Statistics & Probability Letters* **45** 11–22.

Carroll, J Douglas, Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika* **35**(3) 283–319.

Carroll, J Douglas, Sandra Pruzansky, Joseph B Kruskal. 1980. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika* **45**(1) 3–24.

Chaikin, Paul M, Tom C Lubensky. 2000. *Principles of condensed matter physics*, vol. 1. Cambridge Univ Press.

Chan, Louis KC, Josef Lakonishok. 1993. Institutional trades and intraday stock price behavior. *Journal of Financial Economics* **33**(2) 173–199.

Chan, Louis KC, Josef Lakonishok. 1995. The behavior of stock prices around institutional trades. *The Journal of Finance* **50**(4) 1147–1174.

Chen, Scott Shaobing, David L Donoho, Michael A Saunders. 1998. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing* **20**(1) 33–61.

Chib, Siddhartha, Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. *The American Statistician* **49**(4) 327–335.

Choi, Hyeokho, Justin Romberg, Richard Baraniuk, Nick Kingsbury. 2000. Hidden markov tree modeling of complex wavelet transforms. *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 1. IEEE, 133–136.

Churchill, Gary A. 1992. Hidden markov chains and the analysis of genome structure. *Computers & chemistry* **16**(2) 107–115.

Colombo, Marco, Jacek Gondzio, Andreas Grothey. 2011. A warm-start approach for large-scale stochastic linear programs. *Mathematical Programming* **127**(2) 371–397.

Cont, Rama, José Da Fonseca, et al. 2002. Dynamics of implied volatility surfaces. *Quantitative finance* **2**(1) 45–60.

Cowles, Mary Kathryn, Bradley P Carlin. 1996. Markov chain monte carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association* **91**(434) 883–904.

CPLEX, IBM ILOG. 2009. V12. 1: Users manual for cplex. *International Business Machines Corporation* **46**(53) 157.

Cplex, ILOG. 2007. 11.0 users manual. *ILOG SA, Gentilly, France* .

Dai, Yu-Hong, Roger Fletcher. 2005. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. *Numerische Mathematik* **100**(1) 21–47.

Damoiseaux, JS, SARB Rombouts, F Barkhof, P Scheltens, CJ Stam, Stephen M Smith, CF Beckmann. 2006. Consistent resting-state networks across healthy subjects. *Proceedings of the national academy of sciences* **103**(37) 13848–13853.

Danielson, Donald A, DA Danielson. 1997. *Vectors and tensors in engineering and physics*. Addison-Wesley Reading.

d'Aspremont, Alexandre, Francis Bach, Laurent El Ghaoui. 2008. Optimal solutions for sparse principal component analysis. *The Journal of Machine Learning Research* **9** 1269–1294.

D'Aspremont, Alexandre, Laurent El Ghaoui, Michael I Jordan, Gert R G Lanckriet. 2007. A direct formulation for sparse PCA using semidefinite programming. *SIAM review* **49**(3) 434–448.

Davis, Geoff, Stephane Mallat, Marco Avellaneda. 1997. Adaptive greedy approximations. *Constructive approximation* **13**(1) 57–98.

Davis, Timothy A, Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* **38**(1) 1.

De Lathauwer, Lieven, Bart De Moor. 1998. From matrix to tensor: Multilinear algebra and signal processing. *INSTITUTE OF MATHEMATICS AND ITS APPLICATIONS CONFERENCE SERIES*, vol. 67. Citeseer, 1–16.

De Lathauwer, Lieven, Bart De Moor, Joos Vandewalle. 2000a. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications* **21**(4) 1253–1278.

De Lathauwer, Lieven, Bart De Moor, Joos Vandewalle. 2000b. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications* **21**(4) 1324–1342.

De Silva, Vin, Lek-Heng Lim. 2008. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications* **30**(3) 1084–1127.

Dean, J, S Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Communications of the ACM* **51**(1) 107–113.

Edelman, Alan, Tomás A Arias, Steven T Smith. 1998. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications* **20**(2) 303–353.

Elliott, Robert J, Lakhdar Aggoun, John B Moore. 1995. *Hidden Markov Models*. Springer.

Evgeniou, A, Massimiliano Pontil. 2007. Multi-task feature learning. *Advances in neural information processing systems: Proceedings of the 2006 conference*, vol. 19. The MIT Press, 41.

Fan, Jianqing, Fang Han, Han Liu. 2013. Challenges of big data analysis. *arXiv preprint arXiv:1308.1479* .

Fan, Jianqing, Runze Li. 2001. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association* **96**(456) 1348–1360.

Fazel, Maryam, Haitham Hindi, Stephen P Boyd. 2001. A rank minimization heuristic with

application to minimum order system approximation. *American Control Conference, 2001. Proceedings of the 2001*, vol. 6. IEEE, 4734–4739.

Friston, KJ, CD Frith, PF Liddle, RSJ Frackowiak. 1993. Functional connectivity: the principal-component analysis of large (pet) data sets. *Journal of cerebral blood flow and metabolism* **13** 5–5.

Gabay, Daniel, Bertrand Mercier. 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications* **2**(1) 17–40.

Gandy, Silvia, Benjamin Recht, Isao Yamada. 2011. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse Problems* **27**(2) 025010.

Gelman, Andrew, Donald B Rubin. 1992. Inference from iterative simulation using multiple sequences. *Statistical science* 457–472.

Geman, S, D Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (6) 721–741.

Goldsmith, J, J Feder, C M Crainiceanu, B Caffo, D Reich. 2010. Penalized functional regression. *Johns Hopkins University, Dept. of Biostatistics Working Papers* 204.

Goldstein, Tom, Stanley Osher. 2009. The split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences* **2**(2) 323–343.

Grieve, SM, LM Williams, RH Paul, CR Clark, E Gordon. 2007. Cognitive aging, executive function, and fractional anisotropy: a diffusion tensor mr imaging study. *American Journal of Neuroradiology* **28**(2) 226–235.

Hamilton, James D. 1989. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the Econometric Society* 357–384.

Hamilton, James D. 1990. Analysis of time series subject to changes in regime. *Journal of econometrics* **45**(1) 39–70.

Hamilton, James D. 2005. Regime-switching models .

Hancock, Peter JB, A Mike Burton, Vicki Bruce. 1996. Face processing: Human perception and principal components analysis. *Memory & Cognition* **24**(1) 26–40.

Harshman, Richard A. 1970. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis .

Harshman, Richard A. 1972. Parafac2: Mathematical and technical notes. *UCLA working papers in phonetics* **22** 30–44.

Harshman, Richard A. 1978. Models for analysis of asymmetrical relationships among n objects or stimuli. *First Joint Meeting of the Psychometric Society and the Society for Mathematical Psychology, McMaster University, Hamilton, Ontario*.

Harshman, Richard A, Margaret E Lundy. 1996. Uniqueness proof for a family of models sharing features of tucker's three-mode factor analysis and parafac/candecomp. *Psychometrika* **61**(1) 133–154.

Hartley, Richard, Andrew Zisserman. 2003. *Multiple view geometry in computer vision*. Cambridge university press.

Hastie, Trevor, Robert Tibshirani, Jerome Friedman, James Franklin. 2005. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer* **27**(2) 83–85.

Helmke, Uwe, John B Moore, Würzburg Germany. 1994. Optimization and dynamical systems .

Hutter, Frank, Holger H Hoos, Kevin Leyton-Brown. 2010. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 186–202.

Institute, McKinsey Global. 2011. Big data: The next frontierfor innovation, competition,and productivity. \url{http://www.mckinsey.com/mgi/publications/big_data/pdfs/MGI_big_data_full_report.pdf}.

Isard, M, M Budiu, Y Yu, A Birrell, D Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review* **41**(3) 59–72.

Ishwaran, Hemant, Lancelot F James. 2001. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association* **96**(453).

Jeevanjee, Nadir. 2011. *An Introduction to Tensors and Group Theory for Physicists*. Springer.

Jolliffe, Ian T. 1995. Rotation of principal components: choice of normalization constraints. *Journal of Applied Statistics* **22**(1) 29–35.

Jolliffe, Ian T, Nickolay T Trendafilov, Mudassir Uddin. 2003. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics* **12**(3) 531–547.

Jolliffe, IT. 1986. Principal component analysis .

Journée, Michel, Yurii Nesterov, Peter Richtárik, Rodolphe Sepulchre. 2010. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research* **11** 517–553.

Juang, Biing Hwang, Laurence R Rabiner. 1991. Hidden markov models for speech recognition. *Technometrics* **33**(3) 251–272.

Khandania, Amir E, Andrew W Lob. 2007. What happened to the quants in august 2007? *Journal of Investment Management* **5**(4) 5–54.

Kim, Chang-Jin, Charles R Nelson. 1999. State-space models with regime switching: classical and gibbs-sampling approaches with applications. *MIT Press Books* **1**.

Kim, Seung-Jean, Kwangmoo Koh, Michael Lustig, Stephen Boyd, Dimitry Gorinevsky. 2007. An interior-point method for large-scale l 1-regularized least squares. *Selected Topics in Signal Processing, IEEE Journal of* **1**(4) 606–617.

Kodl, Christopher T, Daniel Franc, Fiona S Anderson, Kelvin O Lim, Elizabeth Seaquist. 2007. Diffusion tensor imaging (dti) identifies defects in white matter microstructure in subjects with diabetes. *Diabetes* **56**.

Kolda, Tamara G, Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* **51**(3) 455–500.

Krogh, Anders, Michael Brown, I Saira Mian, Kimmen Sjölander, David Haussler. 1994. Hidden

markov models in computational biology: Applications to protein modeling. *Journal of molecular biology* **235**(5) 1501–1531.

Laloux, Laurent, Pierre Cizeau, Marc Potters, Jean-Philippe Bouchaud. 2000. Random matrix theory and financial correlations. *International Journal of Theoretical and Applied Finance* **3**(03) 391–397.

Lehmann, Bruce N. 1990. Fads, martingales, and market efficiency. *The Quarterly Journal of Economics* **105**(1) 1–28.

Lenth, Russell V. 2001. Some practical guidelines for effective sample size determination. *The American Statistician* **55**(3) 187–193.

Leroux, Brian G, Martin L Puterman. 1992. Maximum-penalized-likelihood estimation for independent and markov-dependent mixture models. *Biometrics* 545–558.

Lim, Lek-Heng, Pierre Comon. 2009. Nonnegative approximations of nonnegative tensors. *Journal of Chemometrics* **23**(7-8) 432–441.

Liu, Ji, Przemyslaw Musialski, Peter Wonka, Jieping Ye. 2013. Tensor completion for estimating missing values in visual data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **35**(1) 208–220.

Liu, Jun S, Andrew F Neuwald, Charles E Lawrence. 1999. Markovian structures in biological sequence alignments. *Journal of the American Statistical Association* **94**(445) 1–15.

Liu, Jun S, Wing Hung Wong, Augustine Kong. 1994. Covariance structure of the gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika* **81**(1) 27–40.

Lo, Andrew W. 1999. The three p's of total risk management. *Financial Analysts Journal* 13–26.

Lo, Andrew W. 2001. Risk management for hedge funds: Introduction and overview. *Financial Analysts Journal* 16–33.

Lo, Andrew W, Archie Craig MacKinlay. 1990. When are contrarian profits due to stock market overreaction? *Review of Financial studies* **3**(2) 175–205.

Lu, Zhaosong, Yong Zhang. 2012. An augmented Lagrangian approach for sparse principal component analysis. *Mathematical programming* **135**(1-2) 149–193.

Martınez-Montes, Eduardo, Pedro A Valdés-Sosa, Fumikazu Miwakeichi, Robin I Goldman, Mark S Cohen. 2004. Concurrent eeg/fmri analysis by multiway partial least squares. *NeuroImage* **22**(3) 1023–1034.

Medioni, Gerard, Sing Bing Kang. 2004. *Emerging topics in computer vision*. Prentice Hall PTR.

Michael, Katina, Keith W Miller. 2013. Big data: New opportunities and new challenges [guest editors' introduction]. *Computer* **46**(6) 22–24.

Miller, Merton H, Jayaram Muthuswamy, Robert E Whaley. 1994. Mean reversion of standard & poor's 500 index basis changes: Arbitrage-induced or statistical illusion? *The Journal of Finance* **49**(2) 479–513.

Misener, Ruth, Christodoulos A Floudas. 2013. Glomiqo: Global mixed-integer quadratic optimizer. *Journal of Global Optimization* **57**(1) 3–50.

Miwakeichi, Fumikazu, Eduardo Martınez-Montes, Pedro A Valdés-Sosa, Nobuaki Nishiyama, Hiroaki Mizuhara, Yoko Yamaguchi. 2004. Decomposing eeg data into space–time–frequency components using parallel factor analysis. *NeuroImage* **22**(3) 1035–1045.

Moghaddam, Baback, Yair Weiss, Shai Avidan. 2006. Generalized spectral bounds for sparse lda. *Proceedings of the 23rd international conference on Machine learning*. ACM, 641–648.

Moghaddam, Bernard, Yair Weiss, Shai Avidan. 2007. Fast pixel/part selection with sparse eigenvectors. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 1–8.

Murphy, Kevin. 2001. An introduction to graphical models. *A Brief Introduction to Graphical Models and Bayesian Networks* **10**.

Neal, Radford M. 1993. Probabilistic inference using markov chain monte carlo methods .

Neumaier, A, E Groeneveld. 1998. Restricted maximum likelihood estimation of covariances in sparse linear models. *Genetics Selection Evolution* **30**(1) 3–26.

Optimization, Gurobi. 2012. Gurobi optimizer reference manual. *URL: http://www. gurobi. com* .

Owren, Brynjulf, Bruno Welfert. 2000. The newton iteration on lie groups. *BIT Numerical Mathematics* **40**(1) 121–145.

Piger, Jeremy. 2009. Econometrics: Models of regime changes. *Encyclopedia of Complexity and Systems Science*. Springer, 2744–2757.

Poterba, James M, Lawrence H Summers. 1988. Mean reversion in stock prices: Evidence and implications. *Journal of financial economics* **22**(1) 27–59.

Qi, Chunhong, Kyle A Gallivan, P-A Absil. 2010. Riemannian bfgs algorithm with applications. *Recent advances in optimization and its applications in engineering*. Springer, 183–192.

Rabiner, Lawrence. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* **77**(2) 257–286.

Rabiner, Lawrence, Biing-Hwang Juang. 1986. An introduction to hidden markov models. *ASSP Magazine, IEEE* **3**(1) 4–16.

Ramsay, J O. 2006. *Functional data analysis*. Wiley Online Library.

Ramsay, J O, B W Silverman. 2002. *Applied functional data analysis: methods and case studies*. Springer Verlag.

Rauhut, Holger, Reinhold Schneider, Z Stojanac. 2013. Low rank tensor recovery via iterative hard thresholding. *Proc. 10th International Conference on Sampling Theory and Applications*.

Recht, Benjamin, Maryam Fazel, Pablo A Parrilo. 2010. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review* **52**(3) 471–501.

Roberts, Gareth O, Adrian FM Smith. 1994. Simple conditions for the convergence of the gibbs sampler and metropolis-hastings algorithms. *Stochastic processes and their applications* **49**(2) 207–216.

Ruppert, D, M P Wand, R J Carroll. 2003. *Semiparametric regression*, vol. 12. Cambridge Univ Pr.

Schölkopf, Bernhard, Alexander Smola, Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation* **10**(5) 1299–1319.

Scholkopf, Bernhard, Alexander Smola, Klaus-Robert Müller. 1999. Kernel principal component analysis. *Advances in kernel methods-support vector learning*. Citeseer.

Scott, Steven L. 2002. Bayesian methods for hidden markov models. *Journal of the American Statistical Association* **97**(457).

Sharpe, William F. 1998. The sharpe ratio .

Shashua, Amnon, Tamir Hazan. 2005. Non-negative tensor factorization with applications to statistics and computer vision. *Proceedings of the 22nd international conference on Machine learning*. ACM, 792–799.

Shen, Haipeng, Jianhua Z Huang. 2008. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis* **99**(6) 1015–1034.

Si, Xiao-Sheng, Wenbin Wang, Chang-Hua Hu, Dong-Hua Zhou. 2011. Remaining useful life estimation–a review on the statistical data driven approaches. *European Journal of Operational Research* **213**(1) 1–14.

Signoretto, Marco, Lieven De Lathauwer, Johan AK Suykens. 2010. Nuclear norms for tensors and their use for convex multilinear estimation. *Submitted to Linear Algebra and Its Applications* **43**.

Smith, Steven T. 1994. Optimization techniques on riemannian manifolds. *Fields institute communications* **3**(3) 113–135.

Smith, Steven Thomas. 2013. Geometric optimization methods for adaptive filtering. *arXiv preprint arXiv:1305.1886* .

Srebro, Nathan, Jason Rennie, Tommi S Jaakkola. 2004. Maximum-margin matrix factorization. *Advances in neural information processing systems*. 1329–1336.

Sriperumbudur, Bharath K, David A Torres, Gert RG Lanckriet. 2007. Sparse eigen methods by dc programming. *Proceedings of the 24th international conference on Machine learning*. ACM, 831–838.

Tan, Aixin, James P Hobert. 2009. Block gibbs sampling for bayesian random effects models with improper priors: Convergence and regeneration. *Journal of Computational and Graphical Statistics* **18**(4).

Tibshirani, Robert. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 267–288.

Tierney, Luke. 1994. Markov chains for exploring posterior distributions. *the Annals of Statistics* 1701–1728.

Tipping, Michael E, Christopher M Bishop. 1999. Mixtures of probabilistic principal component analyzers. *Neural computation* **11**(2) 443–482.

Tomioka, Ryota, Kazuyuki Aihara. 2007. Classifying matrices with a spectral regularization. *Proceedings of the 24th international conference on Machine learning*. ACM, 895–902.

Tomioka, Ryota, Kohei Hayashi, Hisashi Kashima. 2010. Estimation of low-rank tensors via convex optimization. *arXiv preprint arXiv:1010.0789* .

Tucker, Ledyard R. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* **31**(3) 279–311.

Udriste, Constantin. 1994. *Convex functions and optimization methods on Riemannian manifolds*, vol. 297. Springer.

Van Norden, Simon, Robert Vigfusson. 1996. Regime-switching models. Tech. rep., Working Paper 96-3, Bank of Canada.

Var, ID. 1998. Multivariate data analysis. *vectors* **8**(2).

Vidal, Rene, Yi Ma, Shankar Sastry. 2005. Generalized principal component analysis (gpca). *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(12) 1945–1959.

Wen, Zaiwen, Wotao Yin. 2013. A feasible method for optimization with orthogonality constraints. *Mathematical Programming* 1–38.

Westin, C-F, Stephan E Maier, Hatsuho Mamata, Arya Nabavi, Ferenc A Jolesz, Ron Kikinis. 2002. Processing and visualization for diffusion tensor mri. *Medical image analysis* **6**(2) 93–108.

Yao, F, H G Müller. 2010. Functional quadratic regression. *Biometrika* **97**(1) 49–64.

Yoon, Ji W, Simon P Wilson, K Hun Mok. 2010. A highly efficient blocked gibbs sampler reconstruction of multidimensional nmr spectra. *International Conference on Artificial Intelligence and Statistics*. 940–947.

Yu, Shun-Zheng. 2010. Hidden semi-markov models. *Artificial Intelligence* **174**(2) 215–243.

Yu, Y, M Isard, D Fetterly, M Budiu, Ú Erlingsson, P K Gunda, J Currey. 2008. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. *Proceedings of the 8th USENIX conference on Operating systems design and implementation*. USENIX Association, 1–14.

Yuan, Xiao-Tong, Tong Zhang. 2011. Truncated power method for sparse eigenvalue problems. *arXiv preprint arXiv:1112.2679* .

Zhang, Cun-Hui, et al. 2010. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics* **38**(2) 894–942.

Zhang, Hongchao, William W Hager. 2004. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM Journal on Optimization* **14**(4) 1043–1056.

Zhang, Youwei, Laurent El Ghaoui. 2011. Large-scale sparse principal component analysis with application to text data. *NIPS*, vol. 24. 532–539.

Zou, Hui, Trevor Hastie, Robert Tibshirani. 2006. Sparse principal component analysis. *Journal of computational and graphical statistics* **15**(2) 265–286.